Modelling And Verification In Structured Integrated Circuit Design

by

Irene Buchanan

Technical Report 3642

May 1980

Computer Science

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett-Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

The material in this report is the property of Caltech, and is subject to patent and license agreements between Caltech and its sponsors.

Copyright, California Institute of Technology, 1980

MODELLING AND VERIFICATION IN STRUCTURED INTEGRATED CIRCUIT DESIGN

Irene Buchanan

This Thesis has been submitted to
the University of Edinburgh in
partial Fulfillment of the Requirements
for the degree of Ph.D.

1980

ACKNOWLEDGEMENTS

I would like to thank Dr. J. P. Gray for all his help and encouragement. Thanks are also due to Dr. D. J. Rees, Prof. S. Michaelson and W. Laing for their valuable and much appreciated comments on this thesis. In addition I would like to acknowledge the many useful discussions I have enjoyed with the graduate students, industrial associates and faculty in the Department of Computer Science at the California Institute of Technology.

DECLARATION

I declare that this thesis has been composed by myself and that the work is my own.

ABSTRACT

Traditional design tools based on geometric representations do not provide an adequate base from which to construct and verify silicon implementations of complex systems. More comprehensive structural, physical and behavioural descriptions must be developed from appropriate representations. This thesis proposes models which may be used to construct unified and consistent descriptions of the structural, physical and behavioural attributes of a design. It also discusses a method of capturing these descriptions using a textual representation embedded in an object oriented programming language. A range of subsystems have been implemented within a design environment tailored to the proposed models of the design activity. In addition to the typical graphical feedback and mask making oriented output a comprehensive list of verification procedures has been integrated into the system. These include dimensional design rule checking, electrical calculations, connectivity verification and simulation at a number of levels of abstraction.

CONTENTS

1. INTRODUCTION

The objective of this thesis is to show how a Very Large Scale Integrated Circuit (VLSI) design may be described by a set of models such that it may be verified to a high level of confidence before reaching fabrication. This approach depends on the full and consistent description of the design at all hierarchical levels of abstraction from primitive wires and components through to progressively larger blocks. It is argued that a design system must be aware of the structural, physical and behavioural characteristics of the design in order to perform reliable verification procedures directly on the single, unified design description. An object oriented, general purpose, programming language, SIMULA, is used to construct a design environment or system (ICSYS) in which, through the use of procedural modelling, the design description can be built and verified.

Integrated circuits (ICs) are used in a wide range of application areas, from microprocessors to analogue devices, and are implemented in a large number of technologies e.g. emitter coupled logic (ECL), several types of metal-oxide semiconductors (MOS) and silicon on sapphire (SOS). This thesis deals with the modelling and verification of the structured IC design of digital systems and subsystems in n-channel silicon gate MOS

(NMOS) [Mead80]. However, it is believed that this approach is relevant to other design styles and other technologies and even completely different disciplines. By concentrating on the particular design task named it is shown that complete and detailed program models of the primitives and hierarchical levels of the design can be used to describe and verify the physical, structural and behavioural design intent.

Integrated circuits have been designed for the past 20 years, until recently most commonly by hand using mylar and coloured pencils. The need for computer aids arose from the increasing number of transistors that can be packed onto a silicon chip, currently around 100000. The rate of increase in density is expected to remain constant for some time [Moore75] (Figure 1-1) and extrapolates to ICs containing several million transistors within the next few years.

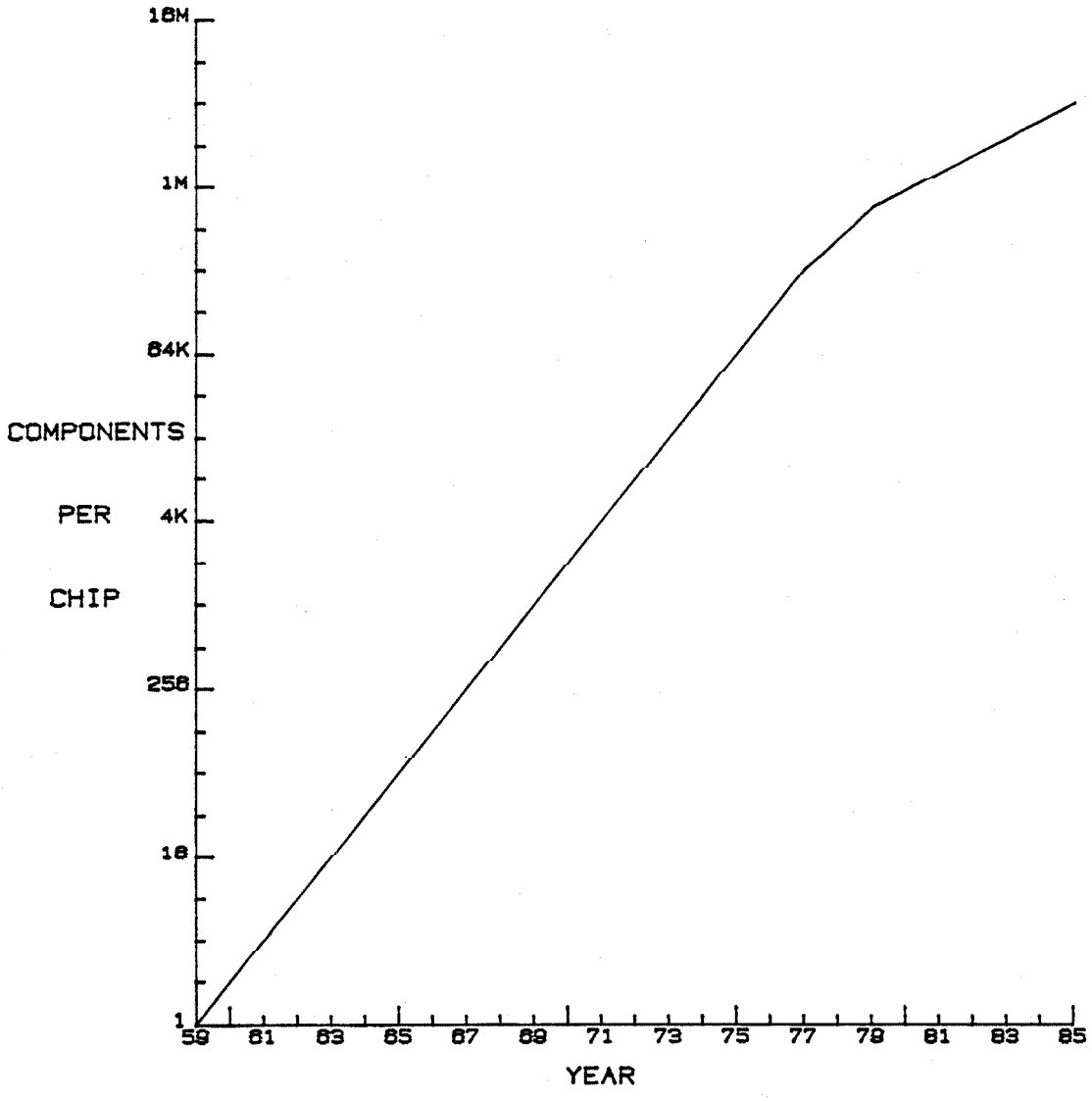


FIGURE 1-1. MOORE'S LAW.

The increasing size and complexity of the design task has necessitated the use of computer aids. They were first employed in digitising, plotting and the production of tapes to describe mask geometries for fabrication. Verification of the design was performed manually by the designer. The size of the problem has made this approach untenable. Design errors result in wasteful fabrication runs, and therefore cost time and money, while more sophisticated computer aids and cheaper computing power have combined to make the use of automatic verification procedures before fabrication the more cost effective solution. It is interesting to note that computer based verification has itself become economically more attractive as a result of the advances made in microelectronics and the subsequent reduction in the cost of computers.

Traditional verification procedures include dimensional and electrical design rule checking and simulation at the circuit, logic and functional level. It will be argued that typical design practice, by concentrating on the physical, or geometric, description of the circuit, does not provide sufficient information for the reliable operation of the required verification procedures; the connectivity of the circuit components and a description of their expected behaviour is also necessary. Simulation illustrates this point most clearly. This argument will

be expanded in Section 1.2. For the purposes of this introduction it is enough to assert that the complete description of an IC design must employ three descriptive domains viz. physical, structural and behavioural.

The thesis also concentrates on a particular design style. Structured design is not a new idea. The much heralded structured programming techniques [Dahl72] are simply one realisation of the generalised philosophy of structured design. Structured design of VLSI may be regarded as its hardware equivalent [Gray79B]. As structured programming evolved to control the complexity of large software systems, so structured design is evolving to control the complexity of VLSI design.

VLSI design exhibits a number of characteristics which make it a different discipline from LSI design. The most crucial design constraint in LSI is packing density; in VLSI it is control of complexity. Naturally, circuit density continues to be important in VLSI design but the partitioning of the design and the management of interconnect are major problems. This is due to their effect on the likelyhood of the design's correctness and the achievement of a reasonable design time. With respect to design aids the movement from LSI to VLSI should see the development of techniques based on structured design principles and a more rigorous approach to verification.

At low levels of integration e.g. MSI and SSI, it is necessary to partition a design over a number of chips. Unfortunately this procedure tends to separate the physical and logical design, complicating the designer's task and the operation of any computerised design aids. It is predicted that VLSI will remove this necessity and that a single silicon surface will be sufficient space in which to implement a reasonably complex function. This technological advance simplifies the relationship between the logical design and the physical design and should lead to improvements in the quality of design aids.

The remainder of this chapter consists of three sections which provide a background for the thesis. The first contains a review of the diversity of manual and automatic techniques employed in the design of integrated circuits and shows how the structured design methodology advocated in this thesis fits into the taxonomy. The second section is devoted to design description and discusses the nature of the structural, physical and behavioural domains in which the design primitives are defined. The last section completes the introduction with some general comments relating to the structured design methodology and IC descriptions.

Traditional computer aided design systems and verification procedures are discussed in Chapter 2. This

provides an historical perspective from which to view the work described in the later chapters.

The control of complexity demands a structured approach to the IC design process and this subject is treated in Chapter 3. Various guidelines are identified and analogies are drawn between structured programming and structured VLSI design.

Chapter 4 describes the set of models chosen for IC design. Primitive components and block structure are obviously necessary but an additional and very important object, the coordinode, is also introduced. It provides the major unifying factor between the structural, physical and behavioural descriptive domains.

Chapter 5 of the thesis progresses to the implementation details of a design system based on the models, methodology and verification procedures proposed in the earlier chapters. The use of the object oriented, general purpose programming language, SIMULA, is discussed and simple design examples are presented.

The features of the design system are further explored in Chapter 6 where they are related to two non-trivial examples, a microprocessor data path and a shift register cell array. Examples are given of the system's

constructional and verification capabilities.

The thesis ends with some conclusions to be drawn from this research, suggests directions for future work and discusses the limitations of the current approach and its implementation.

A number of figures in the text use colour in representing IC designs as either artwork or stick diagrams. The colour assignments to layers are given in the following table.

Layer	Colour
Polysilicon	Red
Diffusion	Green
Metal	Blue
Contacts	Black
Implant	Black

Note that the implant layer is often omitted from plots for the sake of clarity.

1.1 IC DESIGN METHODOLOGIES

Implementations of hardware systems as integrated circuits have been arrived at through a number of design

methodologies. These include custom design, uncommitted logic arrays (ULAs), programmable logic arrays (PLAs), standard cells, read only memories (ROMs), random access memories (RAMs) and structured design. It is useful to classify the various design styles in terms of the degree of regularity of their interconnect paths and the degree of variability of their basic component or cell sizes [Gray79A] (Figure 1-2).

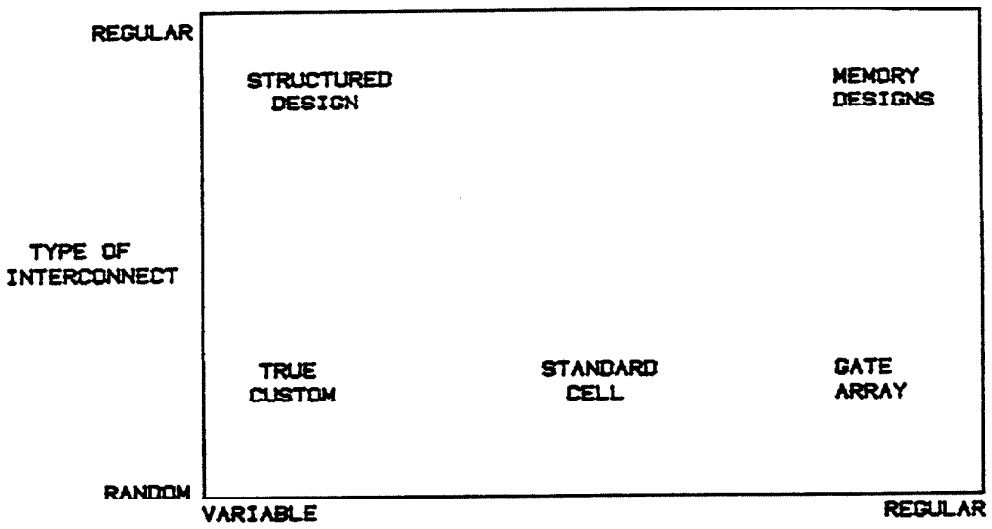


FIGURE 1-2. CLASSIFICATION OF DESIGN STYLES.

In integrated circuits interconnect is a major cost - paths consume silicon area whereas logic virtually "comes for free" [Sutherland77]. Variability in cell size shows to what extent the topology of a design may be reflected in the underlying layout. Since designs include functional blocks of variable size, a design style which could match this variability should be more efficient in silicon area than one which cannot. Thus Figure 1-2 indicates how well each design style is suited to the silicon medium. A description of each methodology justifies its placement in the classification.

Custom design is the process by which a particular function is implemented manually in the minimum silicon area by one or more highly skilled designers. Most ICs currently in production have been designed using this philosophy, from high volume parts e.g. memories and microprocessors to the usually small volume parts designed on contract by a design facility such as the Wolfson Microelectronics Institute (WMI) at the University of Edinburgh. An example of this is the layout of a chip for driving a micrometer counter display (Figure 1-3).

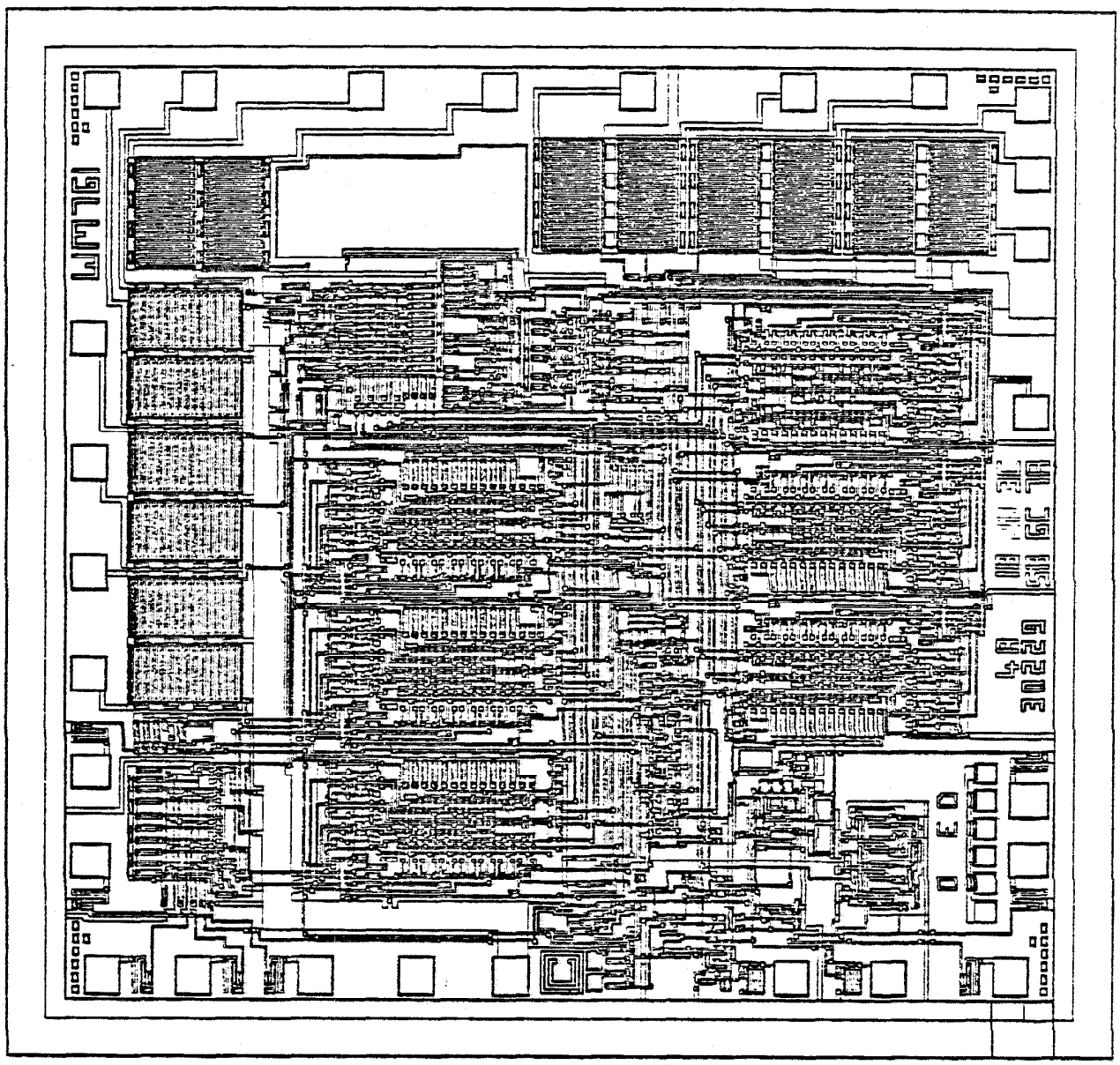


FIGURE 1-3. CUSTOM DESIGN EXAMPLE.*

* REPRODUCED BY PERMISSION OF WMI

The concentration on minimising area is intended to maximise the yield in fabrication since the area of a design and its yield are closely linked [Oldham77, Mead80]. A simple probabilistic model relating yield to chip area generates a Poisson distribution curve and it can be seen that a small increase in area can cause yield to drop sharply [Mead80]. The increasing size and complexity of integrated circuit designs has dictated a need for more regularity so that designs may be better partitioned and verified [Lattin79]. Whether this philosophy will result in a waste of silicon area and resultant reduction in yield is an open question and is of consuming interest to the semiconductor industry.

Gate arrays, also known as uncommitted logic arrays (ULAs), can be most usefully employed when minimising design time is more important than minimising silicon area. This is true for many low to medium volume parts. A ULA consists of clusters of devices which may be locally connected to perform one of several possible logic functions. These clusters are separated by channels through which interconnect paths may be routed e.g. a PMOS gate array designed in WMI (Figure 1-4) [Rose76].

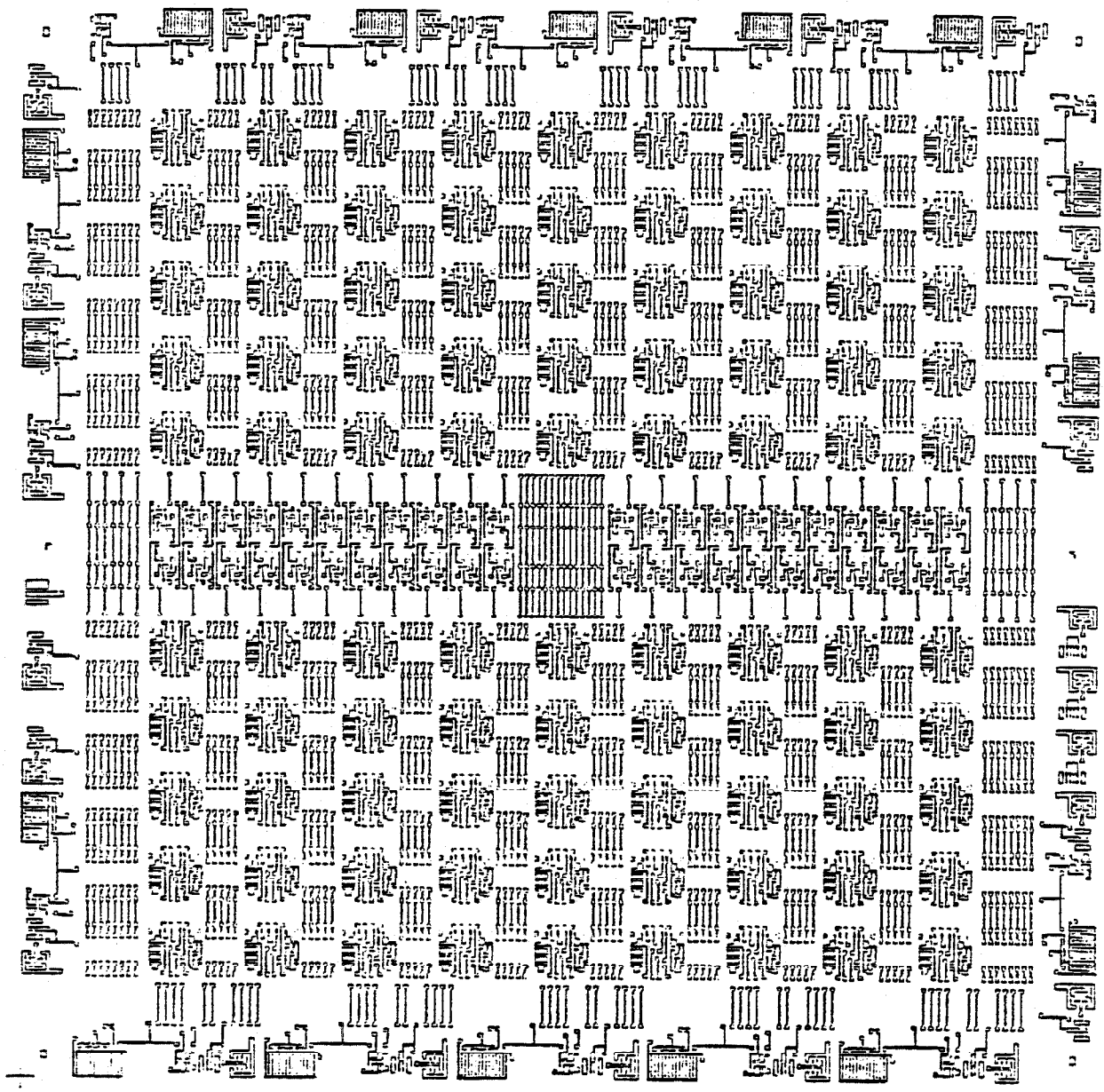


FIGURE 1-4. GATE ARRAY EXAMPLE.*

* REPRODUCED BY PERMISSION OF WMI

Thus although the original design may have included a number of levels of abstraction and been partitioned into several modules at each level, the structure is reduced to a single level of interconnect at the silicon surface. Part of the attraction of this process is that only the interconnect mask, or masks, must be defined for a particular design and since this function can be automated it gives great savings in both design time and mask making costs e.g. in IBM's master slice process [Heller77]. The ratio of circuit density between a structured design and a gate array implementation has been investigated for a small set of chips. The structured designs were found to be more dense by a factor of between 3:1 and 6:1 [Heller79A].

Programmable logic arrays (PLAs) are highly regular structures which can be configured to perform a particular logic function. They may often provide a more efficient implementation in terms of silicon area than the equivalent gate layout. Logic minimisation remains important to reduce the number of product terms and therefore the silicon area and delay associated with a PLA design [Weinberger79]. They may also be generated automatically from logic equations [Ayres79]. These features make PLAs an attractive method of realising some digital subsystem functions. A small PLA implementing a traffic light controller [Mead80] is shown in Figure 1-5.

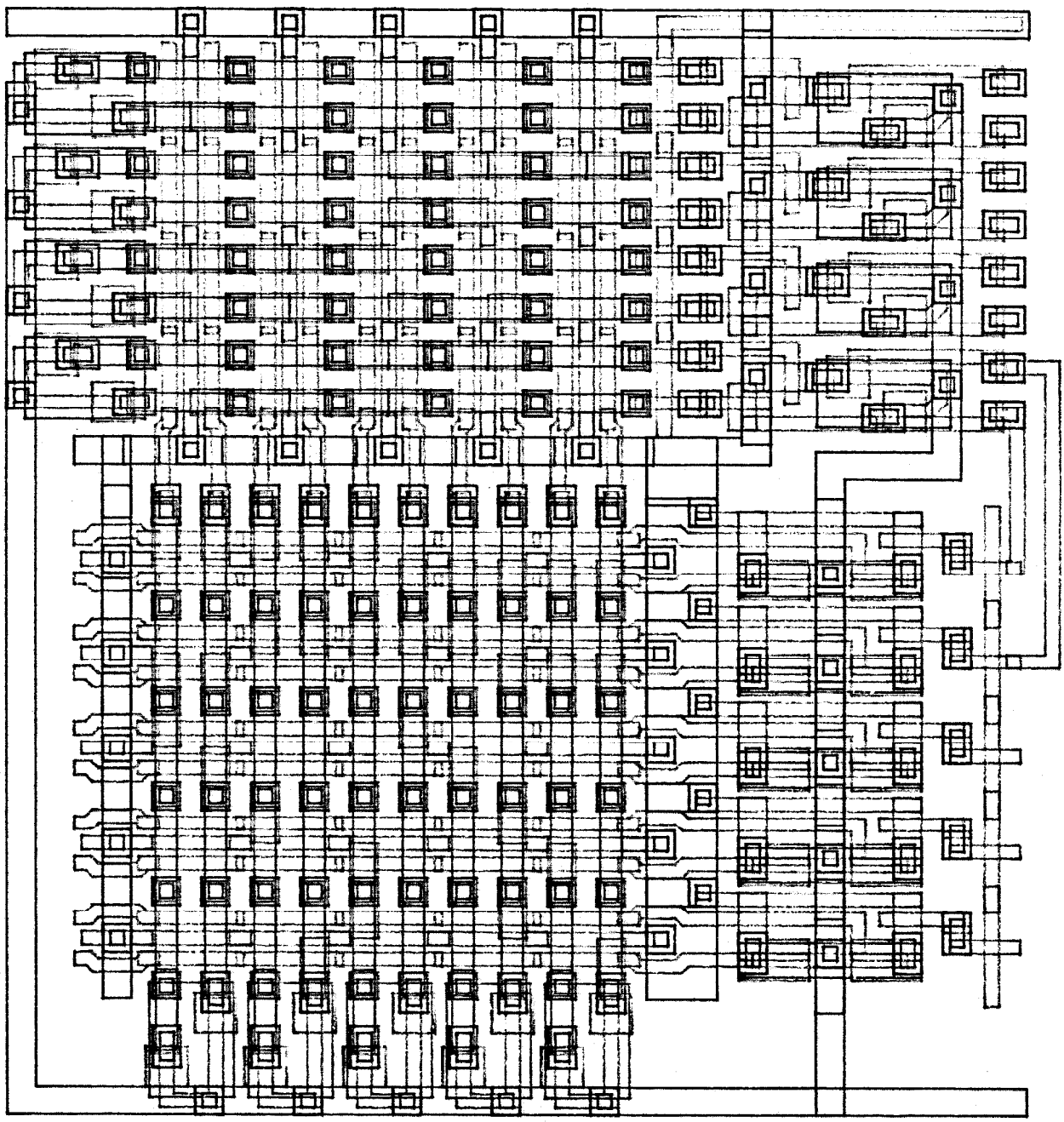


FIGURE 1-5. PLA EXAMPLE.

It was constructed using code written by D. Johannsen under the LAP IC design system [Locanthi78].

The standard cell design technique [Persky76, Preas78] allows a designer to choose cells containing implementations of particular functions from a library and specify interconnections between them. Layout may be manual or automatic. Since these predesigned cells may not perform exactly the function the designer requires and also because regular interconnect cannot in general be achieved, this design style trades flexibility and silicon area for lower design times and correct design at the subcell level. However, the predesigned cells should enable better density to be achieved than does a gate array realisation of the same function. The designer has the additional problem of matching the design hierarchy of a particular problem to that dictated by the cell library. The efficiency of this mapping is a key factor in the applicability of this type of methodology.

Read only memories and random access memories are very highly regular memory structures which may be used to implement logic functions, usually with a high degree of redundancy [Blakeslee79]. Since this redundancy almost always implies a waste of silicon area it is not a technique that is applied as a matter of course.

Structured design [Mead80], as introduced in the last section, emphasises top-down, hierarchical, modular design techniques. Given this general approach some more specific guidelines can be identified for VLSI design. Wiring management is the central problem [Heller79B]. Wiring must be kept regular; random interconnect paths consume silicon area, destroy regularity and make correct design more difficult to achieve. Short paths preserve locality and promote tidy interfacing. Control, data and power/ground are the three logical strands from which the design is created. Assigning each to a preferred mask and direction effectively constrains the overall shape of the design e.g. in NMOS let power/ground run horizontally in metal, let data run horizontally in diffusion and let control run vertically in polysilicon. These assignments lead to designs such as the shift register array in Figure 1-6 [Mead80].

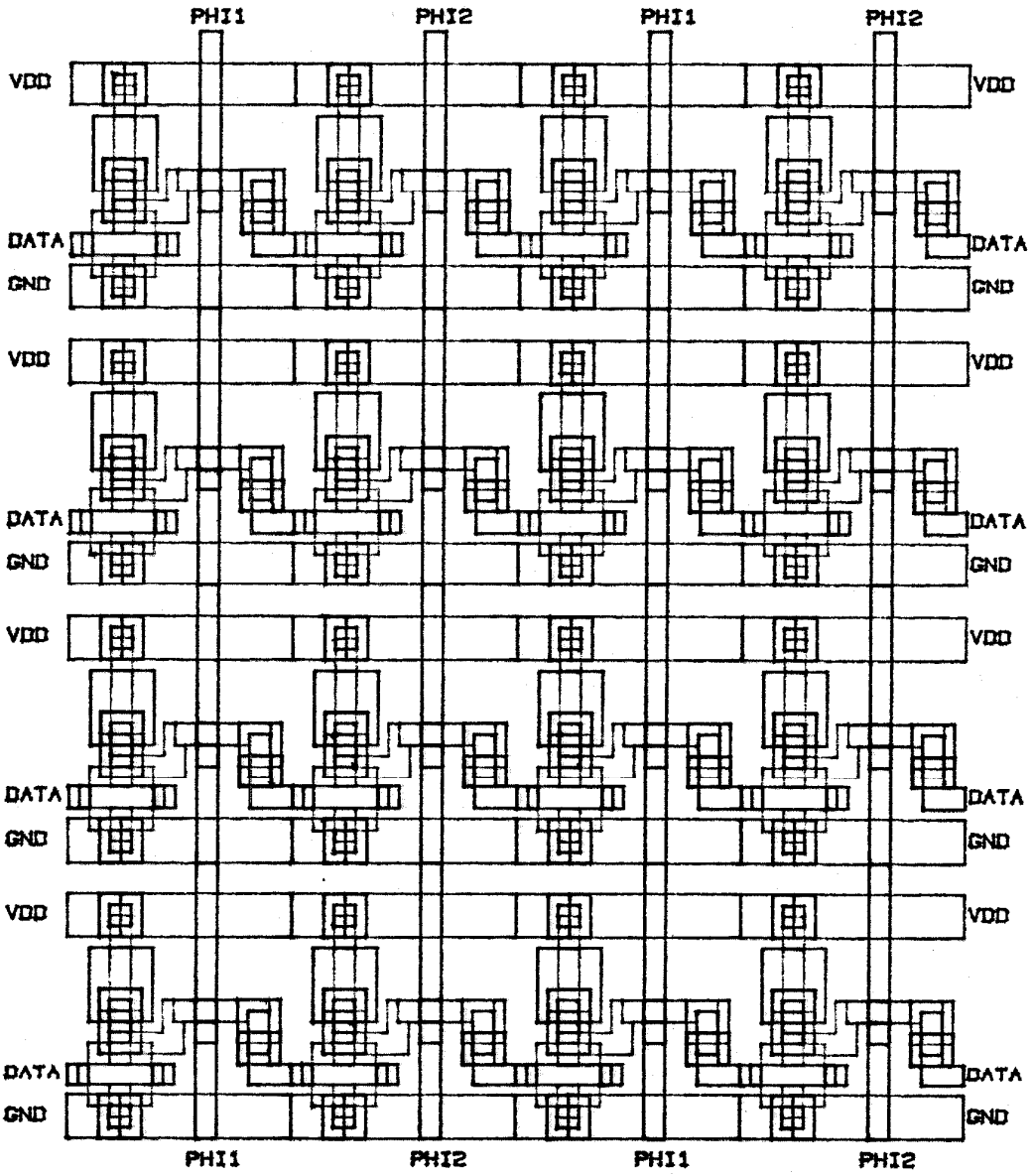


FIGURE 1-6. SHIFT REGISTER ARRAY.

Since transistors are formed where polysilicon crosses diffusion, NMOS structured design typically has a rectangular appearance [Kung80]. A technology with three independent connect layers might have a tendency towards hexagonal cells which can be made to accommodate connections in three separate directions.

Another feature of structured design, also associated with regularity of interconnect, is the specification of cells of identical pitch i.e. of the same physical dimension along their interconnect boundary. This facilitates the design of standardised interconnect schemes e.g. the power, ground and bus connections in the shift register array (Figure 1-6) or the OM2 data path [Mead80] c.f. Section 6.1.2. Thus the traditional placement and routing of the conventional standard cell approach is translated to abutting and stretching in the structured design methodology. Achieving this type of layout might seem to be at the expense of silicon area; all cells must be stretched to fit the largest. However, the absence of tortuous interconnect paths saves area and may overcome the losses at the lower level. Informal estimates put the gain at around 20% over small areas.

The use of arrays and the emphasis on locality and regularity of interconnect makes possible a very direct relationship between the design hierarchy and the physical

hierarchy of implementation in silicon [Gray79A]. Modules are mapped onto unique partitions on the silicon surface. Descending the hierarchy identifies progressively smaller silicon areas and more limited functions.

Finally, the methodology of structured design coupled with the increase in design complexity afforded by increasing circuit density makes it possible to think about novel architectures in terms of silicon layout - the most basic medium of implementation. Parallel processing, by pipelines or arrays, is an obvious example. An identification of the most suitable architecture for a specific algorithm allows the nature of a problem to be investigated to an extent that goes beyond the von Neumann model used at this time in algorithmic complexity studies [Mead80].

One factor that must be taken into account in evaluating each of the design styles and in making comparisons between them is the level of maturity that each style and its associated CAD systems have reached. The completeness of the design methodology and system i.e. its degree of automation and amenability to verification may be regarded as a measure of its maturity. By this criterion the various automated gate array layout systems have advanced further along this route than most design systems. In particular, IBM have included design

guidelines which assure the testability of manufactured parts [Correia77, Eichelberg77]. The increasing complexity of systems causes grave problems in this area and where IBM have led, other design methodologies and systems must undoubtedly follow. PLAs are similarly amenable to automated layout [Ayres79] and can be a useful, speedily generated subsystem of a more primitively designed whole. Printed circuit board (PCB) placement and routing techniques have been developed over a number of years and lend some maturity to both gate array assignment and routing and to standard cell layouts. In the case of structured design computer aids have, until recently, been limited to those available for traditional custom design i.e. graphically based, geometric layout aids [Eades76] or simple graphic languages [Eades76, Sproull80]. The IC design system discussed in this thesis, together with the work at Caltech on silicon compilers and chip assemblers [Johannsen79, Tarolli79] c.f. Section 4.2, attempts to advance the design automation techniques associated with structured design to provide the designer with a set of powerful construction tools capable of dealing with VLSI complexity.

1.2 IC DESIGN DESCRIPTIONS

It is possible to identify three domains of design description viz. structural, physical and behavioural.

Each primitive component and each module at each level in the hierarchy has a description in all three of these domains.

Structurally a design may be described in terms of nets and components, where components may be instances of other blocks or primitive components such as transistors [vanCleemput77]. It may be visualised as boxes or special symbols representing components connected by lines representing nets (Figure 1-7).

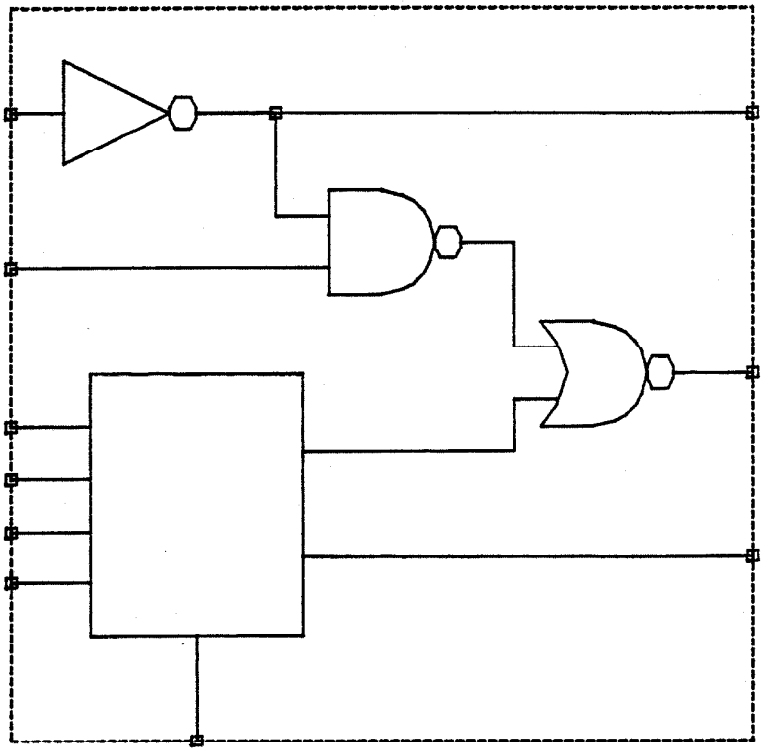


FIGURE 1-7. NETS AND COMPONENTS.

The traditional logic diagram is one such structural description. Many IC designs are constructed from functional blocks which may be implemented as PLAs, ROMs, etc. or may be designed for some specialised task.

Physically an integrated circuit design consists of patterns on particular masks. The defined areas may be represented as boxes, polygons and wires. At a higher level of abstraction, but still within a single IC, a design may be represented as abutting areas on the silicon surface (floor plan), each enclosing a separate module (Figure 1-8). Above the IC level other physical objects appear in the design viz. boards, racks and cabinets. These are part of the physical hierarchy [Bell78].

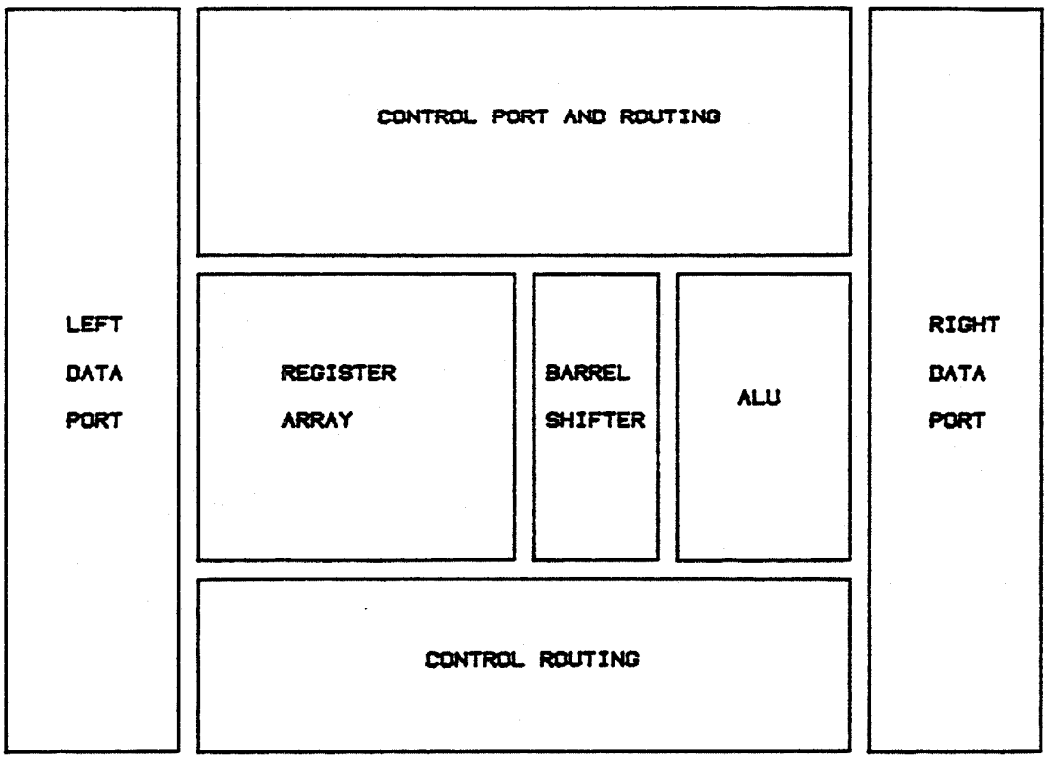


FIGURE 1-8. OM2 FLOOR PLAN.

Behaviourally a design may be described in terms of its function, at the processor memory switch level (PMS) [Bell71] and at the register transfer (RT) levels the instruction set processor notation (ISP) [Barbacci77] could be used, or at a lower level it may be described in terms of logic values on nets or as electrical circuit parameters. Thus a clear hierarchy exists within the behavioural description of a design. Descriptions of behaviour can be represented by special purpose languages e.g. SIMULA [Birtwistle73], DDL [Dietmeyer78] and APL [Iverson62], or timing diagrams, logic equations, etc.

Computer aided design of digital systems seeks to control the mapping between the hierarchies in each descriptive domain. For example, traditional logic design procedures map a logic diagram into TTL packages [McWilliams78, Smith80] on printed circuit or wire wrap boards by assignment, partition, placement and routing. Unifying the separate hierarchies has obvious advantages in the control of the complexity of the overall design task. This factor will be considered in the evaluation of design systems undertaken in Chapter 2.

1.3 SUMMARY

Design is a process of formalising and building models. Different abstractions may be useful at different levels

of the design. IC design may be viewed as the concurrent development of structural, physical and behavioural descriptions. The design process may be handled most effectively by adopting a hierarchical, structured approach. However, designs may be complicated by interference between hierarchies in different descriptive domains e.g. physical packaging may not conform to the structural hierarchy.

Implementation on a single silicon surface can guarantee that VLSI design need not suffer from interference between hierarchies and moreover ensures that the separate hierarchies may be unified within each module of the design description, thereby achieving consistency at all times.

This thesis and the design system it describes contribute to research in the field of IC design aids in a number of ways.

(i) It proposes and implements a view of IC design based on a joint structural, physical and behavioural description.

(ii) It proposes and implements models of design primitives and design modules which allow a designer to describe a circuit at a number of

levels of abstraction. These models also enable a design system to build an integrated design description which is consistent over all three descriptive domains.

(iii) General purpose programming languages are identified as a rich environment in which to implement these design models and the ICSYS system exemplifies one embedding.

(iv) It examines the necessity and illustrates the usefulness and viability of including verification subsystems in the design system which operate from the same data description as those procedures concerned with graphical feedback and mask making.

(v) It identifies structured design of VLSI as an area amenable to powerful and productive design aids and indicates some possible future directions for research.

2. DESIGN SYSTEMS AND VERIFICATION

Design systems provide their users with representations in which to describe their designs. The characteristics of the representation e.g. whether it encompasses the structural, physical and behavioural descriptive domains, dictate the scope of the verification procedures which may be performed on the design by the design system. This chapter explores the development of IC design systems and the verification procedures provided by them within the context of custom design for digital systems.

2.1 IC DESIGN SYSTEMS

The three domains of IC description i.e. the structural, physical and behavioural, have been identified in Section 1.2. Representaions of IC designs are those textual or graphical constructions which the user may employ to convey the design intent to the design system. A classification of representations and their relationship to design descriptions is given in the following table.

Representations	Design Descriptions		
	Physical	Structural	Behavioural
PMS Level	-	Yes	Yes
RT Level	-	Yes	Yes
Schematic	-	Yes	Yes
Stick Diagrams	Semi	Yes	-
Artwork	Yes	-	-

For example, a design system such as GAELIC [Eades76] represents the IC design as boxes, polygons and tracks on masks, and therefore only allows a physical description of the design. Stick diagram based systems e.g. STICKS [Williams77] represent the IC design as coloured wires and symbols (Figure 2-1) i.e. a structural and semi-physical description of the design; the full physical realisation of the circuit is produced automatically.

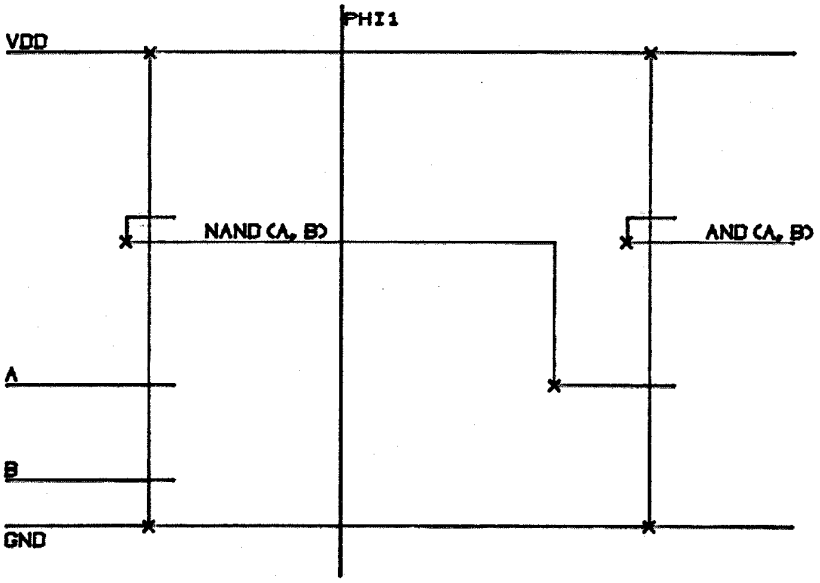


FIGURE 2-1. STICK DIAGRAM.

The first use of computers in IC design occurred in digitising and plotting systems which produced data, on paper or magnetic tape, to drive a mask making machine [Eades76]. A designer's artwork was digitised, stored and plotted for manual checking. Any errors meant redigitising at least part of the design.

A second generation of design systems arrived when interactive graphics allowed these systems to develop on-line editing facilities, thus decreasing design times. There are many examples of commercially available systems which use this approach e.g. APPLICON, CALMA and GAELIC [Eades76]. Some systems also use textual representations and define special purpose languages to describe the physical layout of the design. An interesting early effort in the standardisation of such a language was CAMP [Wood69] from which the GAELIC language developed.

These physically oriented design systems capture only the result of the last phase of the design process, the final translation from logical (structural) design to geometric (physical) layout. This manual process is both time consuming and error prone. Furthermore, in systems which allow independent entry of the logical design it results in the system storing separate structural and physical descriptions of a design and these may be inconsistent.

At this point it is instructive to examine more closely one particular second generation design system. The GAELIC system [Eades76] was developed in the Wolfson Microelectronics Institute and the Department of Computer Science at the University of Edinburgh and is now marketed by Compeda Ltd. It is chosen to be representative of current, commercially available, CAD systems for IC design. The system consists of a suite of programs operating on a central data file (Figure 2-2). The data structure is organised hierarchically, allowing cells to be instanced from other cells and employing rings of data beads to store the description (Figure 2-3). It contains only the geometric description of the design.

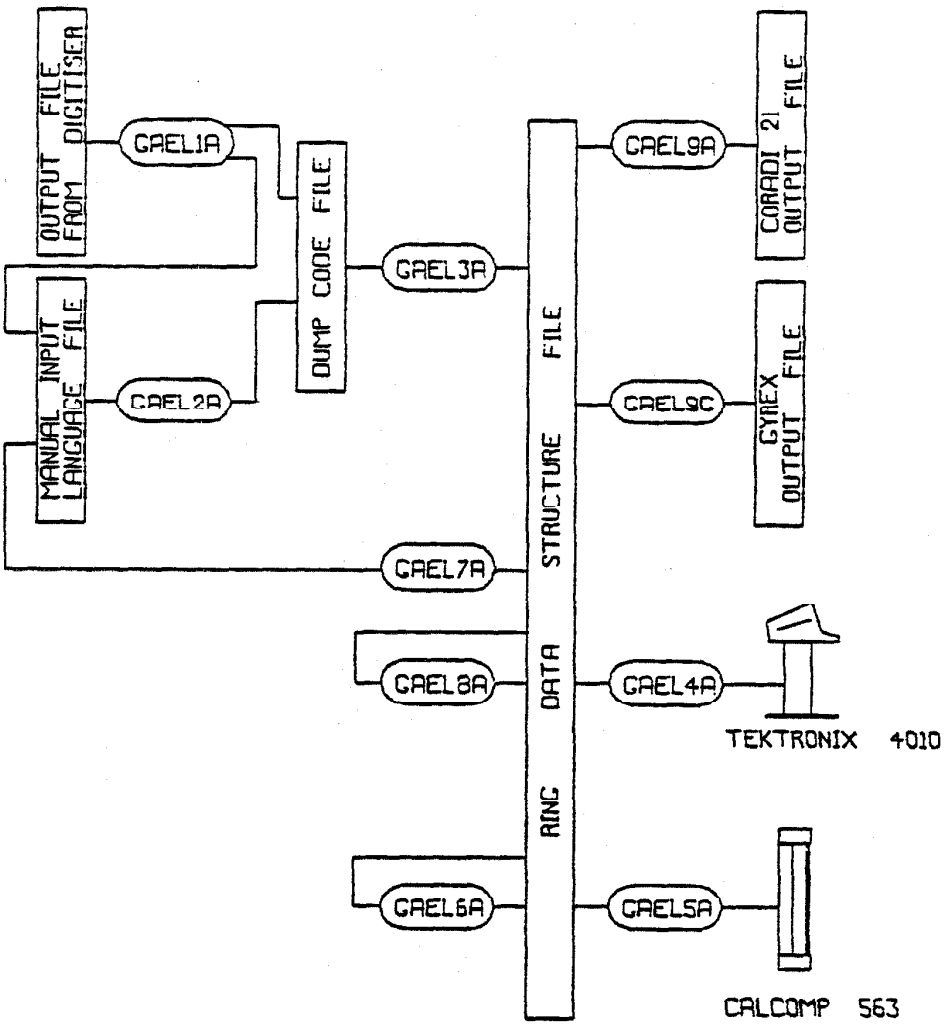


FIGURE 2-2. GAELIC SYSTEM.*

* REPRODUCED BY PERMISSION OF DR. J. EADES

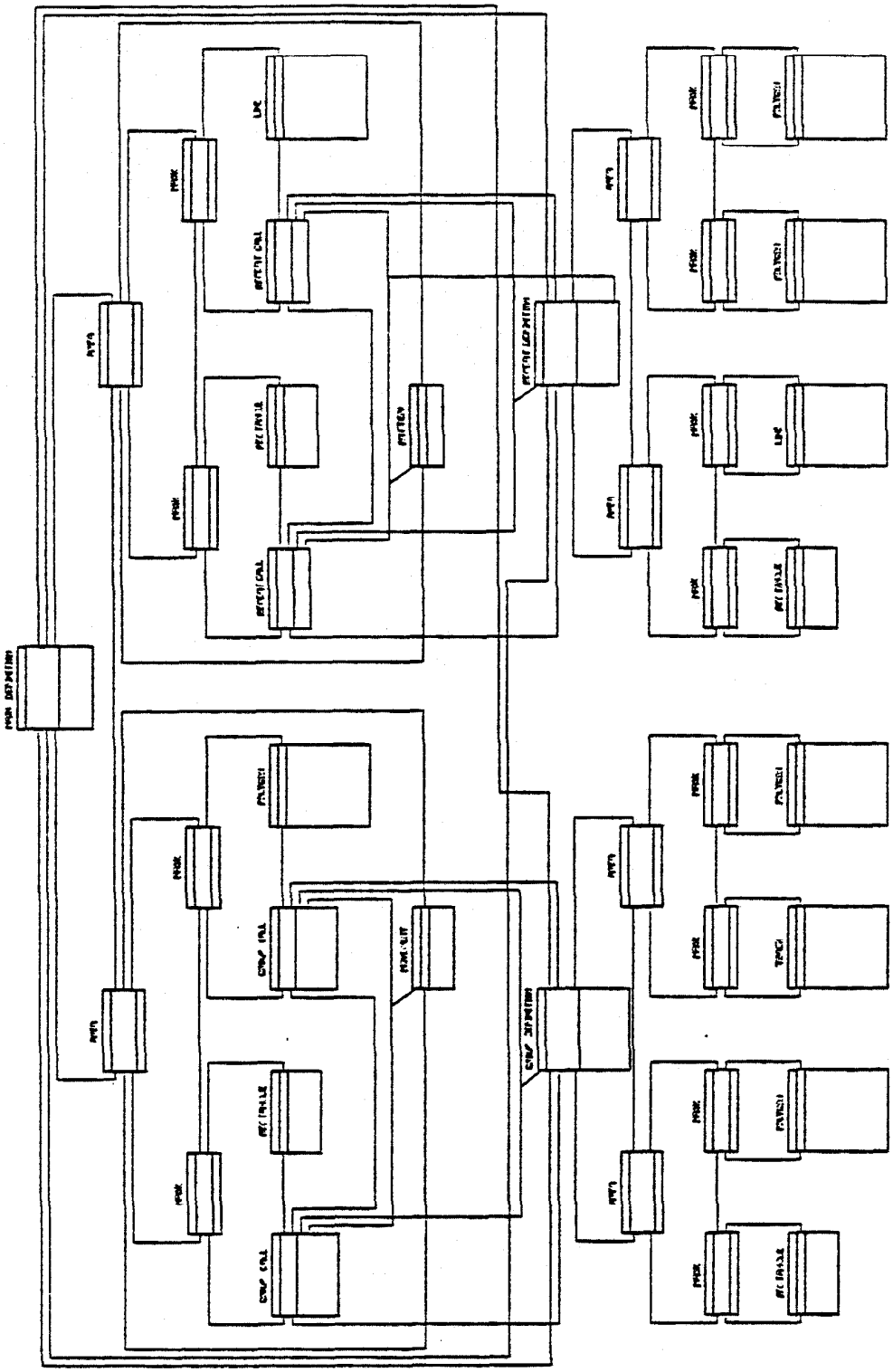


FIGURE 2-3. GAELIC DATA STRUCTURE.*

* REPRODUCED BY PERMISSION OF DR. J. EADES

Digitising, interactive editing and the output of pattern generation tapes are all catered for in a straightforward design process with suitable feedback loops. Verification procedures, though also working from the same central data file, appear as added extras which were bolted on as the need for each became apparent. Computer aided verification only became essential with the increased complexity of relatively recent designs. The problem is that the geometrically based data file does not contain all the information the verification programs need to perform reliably (c.f. Section 2.2). Unfortunately neither the textual nor graphical forms of design input in the GAELIC suite are capable of dealing with a more complete description.

Symbolic layout i.e. the use of particular characters to define the presence of a layer or combination of layers at a grid point, has been a successful tool in at least one industrial environment [Gibson76], though the system has been critised both on grounds of human factors and for wasting silicon area. Characters on lineprinter output are identified with mask overlays. Each character is regarded as occupying an area of the silicon surface, typically of minimum device dimensions. Designs of any size require sections of output to be taped together.

A second generation of symbolic systems can be

identified in systems based on the stick diagram representation (Figure 2-1) which are under development [Williams77, Dunlop78, Hseuh79]. Such systems cater for an accurate structural description and allow some guidance on physical constraints e.g. assignment of wires to layers and relative positions of devices. From this information a physical layout can be produced automatically but satisfactory density has proved difficult to achieve. A non-trivial design example has still to be published.

One advantage of a system based on stick diagrams is that dimensional design rule checking is redundant since the physical layout is synthesised from the structural description according to the design rules built into the design system; this is an example of the increasingly important design philosophy of correctness by construction i.e. the use of algorithms to synthesise layout as opposed to the use of algorithms to analyse and verify layout.

It should also be apparent that changes to design rules may be reflected in the physical layout by rerunning the synthesis program and do not require lengthy reworking of the layout. However, heuristics can often be improved on by human effort and, in the quest for maximum density, designers also find it desirable to have intimate control over some parts of the layout and may wish to include special purpose artwork. Since such geometry by-passes

the synthesis procedure it invalidates the design rule correctness assertion.

Thus a major problem in a stick based design system is to provide the designer with sufficient flexibility without compromising the virtue of the design method. It is difficult at this time to see a solution to this problem given the lack of maturity of this field of research.

2.2 IC DESIGN VERIFICATION

The cost of the design cycle i.e. design, fabrication, testing, redesign, etc, is considerable both in terms of time and money. Designers of small scale and medium scale integrated circuits (SSI & MSI) had some chance of discovering all errors by manual inspection but designers of large scale and very large scale integrated circuits (LSI & VLSI) do not. Thus the increasing size and complexity of IC designs has led to the developmemt and use of computer aids, firstly in verification procedures and secondly in design systems based on correctness by construction.

This section enumerates the various verification procedures associated with IC design. Each subsection deals with a single procedure, explains the necessity for

its existence and discusses the problems encountered in its implementation. Where appropriate the section contains information on the characteristics of such programs as are reported in the literature or available commercially. Taken as a whole the section argues that complete and reliable verification procedures can only operate within a system which includes integrated structural, physical and behavioural descriptions.

2.2.1 DIMENSIONAL DESIGN RULE CHECKING

Dimensional design rule checking must be performed on manually produced artwork to ensure that it conforms to rules laid down by the management of the processing line. These rules are intended to guide the designer in laying out the circuit such that even when processing is at its worst the circuit topology is preserved [McGrath79]. Therefore the rules relate to widths of tracks and separations between shapes on the same or different layers e.g. polysilicon to diffusion separation (Figure 2-4(a)) and metal track widths (Figure 2-4(b)) in NMOS [Mead80]. Simple layer to layer rules can be prescribed solely on the physical or geometric data. More complex rules e.g. transistor to contact cut separation (Figure 2-4(c)) require either previous knowledge of the logical structure of the design or geometric analysis to find the structure e.g. a transistor, prior to the DRC operation.

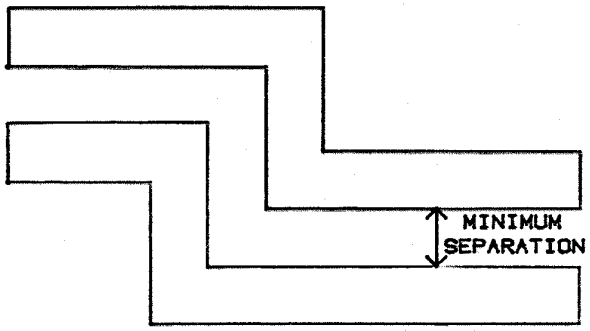


FIGURE 2-4(A). POLYSILICON TO DIFFUSION SEPARATION.

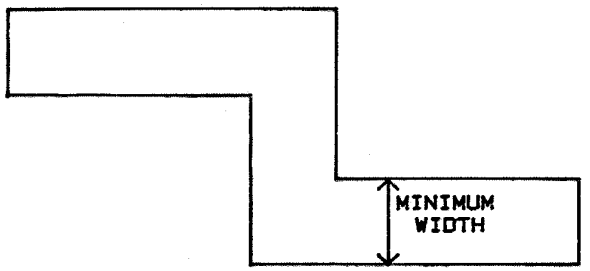


FIGURE 2-4(B). METAL WIDTH.

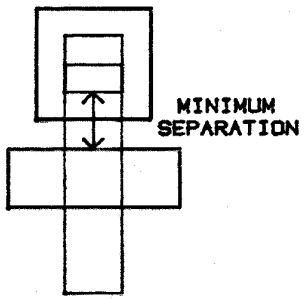


FIGURE 2-4(C). CONTACT TO TRANSISTOR SEPARATION.

Since this geometric analysis is being performed on data that may contain errors it is inherently an unreliable process. Errors may propagate through to the design rule checking stage and result in spurious or misleading design rule violation flags and may allow genuine errors to remain undetected.

Transistors have not been assigned structural significance by the designer; they are formed by overlaying artwork and must first be deduced by geometric analysis. Thus a design error may produce a spurious transistor and undermine the first stage of the design rule check. Checking procedures based on such erroneous constructions further compound the verification problem i.e. it is essential that all design rule violations are found and that no spurious violations are flagged. The first is an obvious necessity since overlooked violations may cause an extra design iteration or may reduce yield in production. That no spurious design violations are flagged is almost as important since they serve as a distraction to the designer who may, in error, dismiss a genuine violation as spurious if the procedure is known to be unreliable. The conclusion to be reached is that a dimensional design rule checking system must be aware of both the structural and physical descriptions of the design in order to provide a reliable service to the designer.

There is also another problem in that the procedure for checking design rules requires manipulation of the layout geometries. Since these are typically very complex, comprising perhaps 100,000 shapes, this is a computationally lengthy process. Various sorting methods, including the moving windows technique [Wilcox78] and the construction of lists of shapes ordered in the direction of the x or y axis [Preas76, McCaw79], have been employed in attempts to reduce the execution time of dimensional design rule checkers [Baird77]. Many checking programs use a polygon package to perform the detailed geometric analysis [Sutherland78, Barton80]. The original design rule checking program in the GAELIC suite, subsequently developed further by Compeda Ltd., was produced by the author and uses this method. An important point to note is that such a package must use circular arc inflation and deflation strategies to avoid the inside and outside corners pathologies. For example, a spurious minimum width violation is generated if the rule is tested by deflating the shape in Figure 2-5(a) by half the desired minimum width using straight line at vertices. The existence of an overlap area is detected and reported as a violation but this is erroneous since the distance in question i.e. between the two inside corners, is greater than the minimum permitted width. Circular arc deflation avoids these spurious violations as shown in Figure 2-5(b).

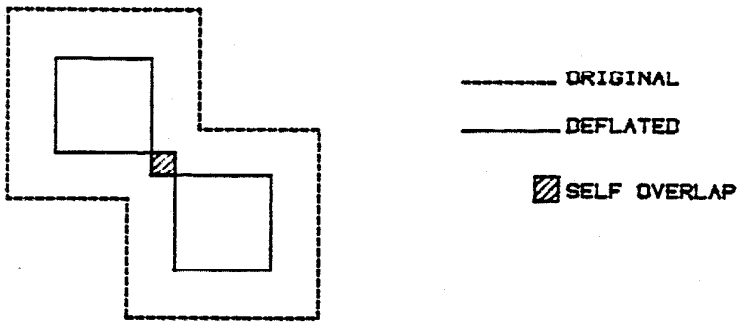


FIGURE 2-5(A). MINIMUM WIDTH - SPURIOUS VIOLATION.

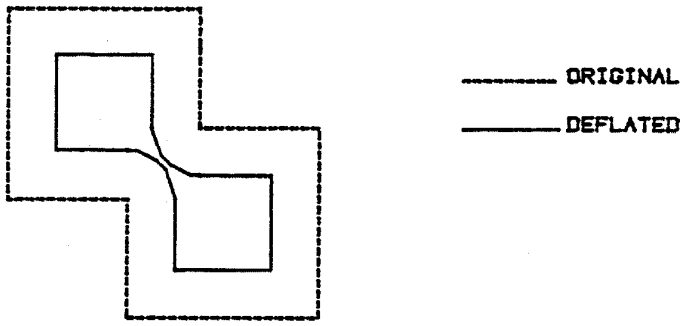


FIGURE 2-5(B). MINIMUM WIDTH - NO VIOLATION.

A similar example involving the minimum separation check and the outside corners of shapes is shown in Figures 2-6(a) and 2-6(b).

The complexity of the dimensional design rule checking procedure has been a topic of concern for some time [Baird77]. If a complete circuit, consisting of hundreds of thousands of data items, must be checked as a whole then even minor improvements in the complexity of the operation will give substantial gains in performance. However, if the design can be partitioned hierarchically into non-interfering modules that can be checked separately then even greater performance gains can be achieved.

Several systems use a technology independent special purpose language to describe the design rules for a particular process e.g. GAELIC (Figure 2-7). This is an advantage since it reduces the amount of new coding involved in changing the design rules or in modifying the design system for a different process.

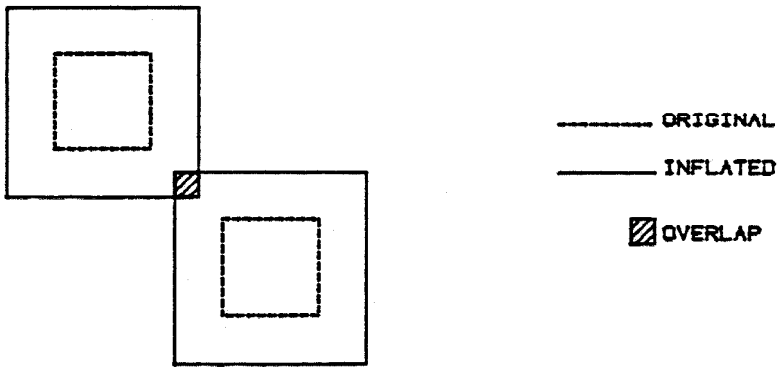


FIGURE 2-8(A). MINIMUM SEPARATION - SPURIOUS VIOLATION.

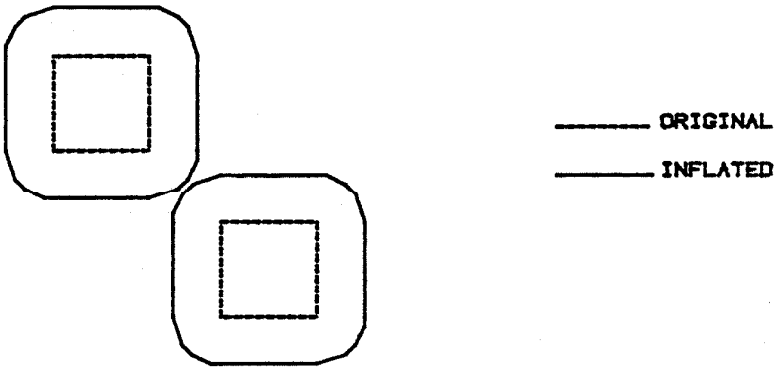
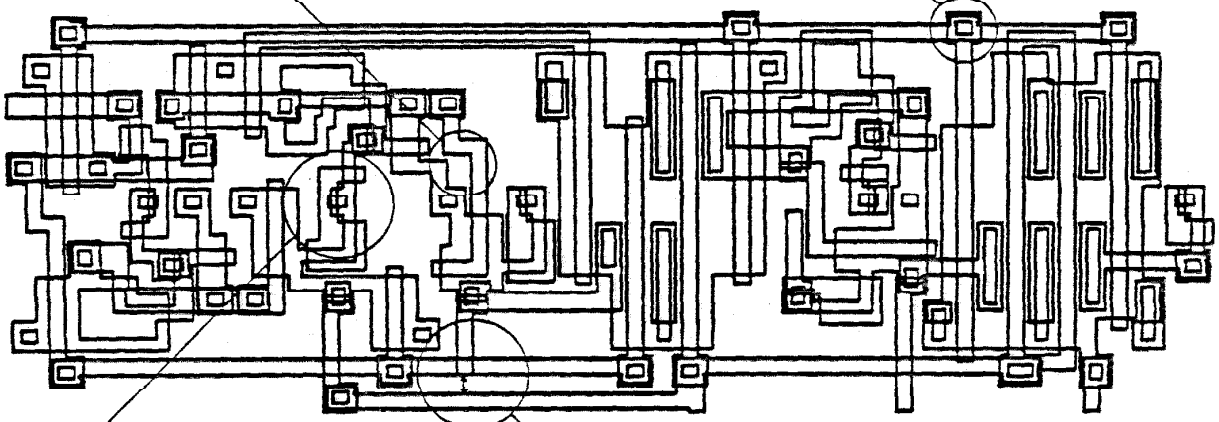


FIGURE 2-8(B). MINIMUM SEPARATION - NO VIOLATION.

```
RULE MOS 1
! SIMPLE SEPARATION TEST
S1 IS POLY MASK 1
S2 IS POLY MASK 2
FAIL "SEPARATION ERROR"
IF SPACING (S1, S2) <CONST
END
```

```
RULE MOS 2
! ENCLOSED SPACING TEST
S1 IS POLY MASK 4
S2 IS RECT MASK 3
FAIL "CLEARANCE ERROR"
IF CLEARANCE (S1, S2) <CONST
END
```



```
RULE MOS 3
! INTERLIMB SPACING TEST
S1 IS POLY MASK 2
FAIL "INTERLIMB ERROR"
IF INTERLIMB (S1) <CONST
END
```

```
RULE MOS 4
! METAL SPACING NEAR BUTTED POLY
M1, M2 IS POLY MASK 4
P IS POLY, RECT MASK 2
FAIL "METAL SPACING" IF PARTED (M1, P)
AND NOT PARTED (P, M2)
AND SPACING (M1, M2) <CONST
END
```

(KEYWORDS ARE SHOWN UNDERLINED)

FIGURE 2-7. DESIGN RULE LANGUAGE.*

* REPRODUCED BY PERMISSION OF COMPEDA LTD.

2.2.2 ELECTRICAL CONSIDERATIONS

In order to construct correctly sized devices a designer requires feedback about the electrical properties of both devices and their connecting paths. Capacitances, resistances and pullup/pulldown ratios are all examples of calculations which an IC design system should be capable of making but which in general are left for the designer to perform manually. To provide these facilities the design system must be aware of the structural, physical and behavioural descriptions of the design.

2.2.3 CONNECTIVITY

Connectivity errors fall into two categories; the absence of a necessary connection and the presence of an unintended connection. For example, if an inter-layer contact is omitted then no connection is made, whereas if two wires in the same layer touch or cross in error then an erroneous extra connection is produced.

An IC design system which contains only a physical description must use geometric analysis to produce a connectivity list. Nets and components may be recognised by procedures similar to those mentioned in Section 2.2.1 with the same reservations regarding reliability. Several systems have been reported which perform this analysis

[Dobes76, Russell78]. Technology independence appears to be more difficult to achieve than for dimensional design rule checking.

The next stage in the verification process is to take the output from the analysis and match it to an independently constructed logic or circuit diagram translated to a machine readable form. The output from a program to extract logical connectivity can be compared against this logical description and discrepancies can be investigated as errors.

Unfortunately this matching process is not a simple task. It is essentially the graph isomorphism problem, which is known to be NP-complete [Aho74]. However, the problem may be simplified by a consistent labelling of nodes in each descriptive domain, thus partitioning the problem into subproblems which may be solved within a reasonable time. A case of engineers rushing in where mathematicians fear to tread? This method of tackling connectivity verification is used by at least one major semiconductor manufacturer but, unfortunately, this work has not been published, presumably for commercial reasons.

2.2.4 SIMULATION

Simulation is a vitally important tool in ensuring design correctness. In fact it is the only tool that is

currently available. Some general analogies are possible between hardware design verification and program verification. It is to the detriment of physically based systems, which make up the bulk of the IC design systems currently in use, that any simulation procedures do not work directly from the complete structural and physical design description. Since simulation requires components and connectivity it must be performed on other independently entered and, it is to be hoped, consistent data. Unfortunately, this is not usually proven to be the case. Therefore a design system may only simulate a design with assured reliability in the presence of consistent descriptions of the IC in all three descriptive domains.

Simulation may be employed at a number of different levels of abstraction e.g. processor memory switch (PMS) [Bell71], register transfer (RT) [Bell72], logic design [Lewin70] and electrical circuit [Nagel75].

It is also desirable, but difficult, to enable the designer to mix these levels allowing concentration on a small part of a large design within its working context without incurring the computation times associated with the simulation of a whole chip. A number of problems may be identified in connection with multilevel simulators. These are related to interfacing between different levels,

controlling the volume of input and output information, presenting that information in a way that is meaningful to the designer and ensuring consistency between descriptions of the behaviour of cells at different levels of abstraction.

2.3 SUMMARY

It may be seen from the preceding discussion of verification procedures that a requirement for reliable verification is that the design system must capture a unified and consistent description of the IC design across all three of the descriptive domains. This must take place concurrently and at every hierarchical level of abstraction within the design. Every type of verification procedure requires access to more than one descriptive domain; some require access to all three.

3. STRUCTURED IC DESIGN

As has already been observed in previous chapters the principal reason for adopting a structured design methodology arises from the need to control the increasing design complexity made possible by the technological progress in IC fabrication densities. In this chapter the subject of structured IC design is explored more fully in order to clarify the meaning of the term and to illustrate its relevance to current and future design environments. The presence of a number of analogies between structured IC design and structured programming has been remarked upon previously and is discussed further in the following sections. These analogies relate principally to the abstract view of design and its realisation as code or layout. A particularly interesting and useful summary of structured programming techniques appears in a recent thesis by G. J. Holzmann [Holzmann79].

By exposing the methodology of structured design it is intended that this chapter should provide a set of design guidelines to which IC designers would adhere and which IC design systems would embody. Guidelines are not inviolate; they may be ignored or departed from at the designer's risk. This is a necessary feature of any realistic and flexible design environment.

3.1 BASIC PRINCIPLES

Each of the following subsections deals with a particular aspect of structured design. The advantages of keeping to each guideline are discussed and any analogies with structured programming are presented. The program is viewed as a static embodiment of an algorithm and is compared with the spatial representation of a design.

3.1.1 MODULARITY

The recent growth of interest in structured design principles applied to ICs has led to a number of chip implementations which clearly show that designs partition naturally into modules with well defined boundaries. This does not supply a general, formal proof that every design can be modularised in this fashion but lends credence to the hardware and software engineering solution based on this philosophy.

The advantages of modularity can be observed throughout the whole process of design. The complexity of VLSI design tends to make it economic to partition the work among a number of designers. Each must be presented with one or more viable modules which have well-defined functions and interfaces. In this way designers can work independently of each other with some confidence in the

correctness of the final combined product. The ability to partition a design among several designers without incurring a large organisational and verification overhead is an important step in solving the problem of increasing design times associated with VLSI. Design times are lengthening at the same rate as design density is increasing and, with a shortage of personnel, the problems of industry are acute [Moore79]. It should be noted that the same approach to design is equally valid when only a single designer is involved. The use of modularity is a powerful tool in the control of complexity, both in the mind of the designer and in the organisation and encoding of the design system.

Looking into structured programming at a more detailed level, an important restriction is that the programmer is requested to use only three basic control-flow structures viz. concatenation, conditional selection and iteration. It is interesting to note that structured design of VLSI proposes very similar restrictions. Abutment of blocks is an important way of managing interconnect; this may be equated to concatenation. Programmable structures, such as PLAs and ROMs, and conditional structures, such as the memory and carry stages of the OM2 data path c.f. Section 6.1.2 are analogous to the conditional selection mechanism of structured programming. The regularity of one and two dimensional arrays of cells is a very important feature of

structured design and can be compared with the programmed iteration.

Design verification problems are also eased by partitioning the design into functional modules. Since geometric verification procedures are of at least O(nlogn) complexity due to their embedded sorting algorithms it follows that reducing the amount of data to be verified at each stage results in faster overall verification. The same argument applies to simulation because abstraction of function behind well defined interfaces simplifies computation.

3.1.2 HIERARCHY

The modularity of design is not limited to a single level but extends over a design hierarchy. Different levels of the hierarchy correspond to different granularity of function. Partitioning via modularity and hierarchy allows a well controlled development and structuring of the design. By employing verification procedures at a number of levels in the design hierarchy the correctness of the design can be ascertained with greater reliability, accuracy, control and efficiency than with a fully expanded layout. Structured layouts of digital systems tend to exhibit a very high degree of duplication of submodules. Taking advantage of this

feature within a design hierarchy contributes to a large reduction in the volume of data in the design file. Four or five levels of abstraction have been found in current designs and even this shallow hierarchy yields considerable reduction in the volume of data since the branching ratio at each level is high [Heller79A, Leyking79].

An important feature of structured programming is that larger structures can, and should, be built from the three basic control structures viz. concatenation, conditional selection and iteration. These larger structures may then be used at the next higher level as basic modules. In this way a program hierarchy is constructed. Clearly, exactly the same principles apply to structured design of VLSI. Progressively larger blocks may be built as the constructional tools of abutment, conditional inclusion and arrays are employed.

3.1.3 REGULARITY

A regular design has a number of identifiable characteristics which together describe a style of layout suitable for use in VLSI design. A 2D surface and two independent layers of interconnect tend to favour the solution of the wiring management problem by rectangular tiling.

The high level of repetition of basic modules, especially in both one and two dimensional arrays, has obvious functional benefit and leads to a dense, comprehensible and inductively correct design. It is a relatively simple matter, since modules are of identical size, to ensure that basic building blocks will fit together when instanced in formation. Joining cells by abuttal is a valuable construction mechanism since it saves space and simplifies the wiring management problem.

This style of inter-block connection can be extended to include a variety of cells. There remains a requirement that cells be identically pitched and have connection points at abutting locations but the blocks may have completely different internal organisations. This does not mean that the designer must be tied to strict dimensions imposed on all blocks in a certain set which may be tied together. It has been shown that computer aids can help the designer to construct stretchable cells which, by conforming to the same basic architecture, can be parameterised to fit together [Johannsen79]. Connection points must, of course, continue to be assigned to abutting locations.

Such tessellated organisations are commonly made from rectangular tiles in NMOS technology. This is because NMOS transistors are formed by polysilicon/diffusion

crossings and therefore it is usual to run control and data orthogonally on separate layers. It might be surmised that different technologies or algorithms which prefer a different interconnect strategy might have a preference for another tile shape e.g. hexagonal [Guibas79, Kung80]. However, most technologies provide at least two layers of interconnect and so there is a broad uniformity over technologies of applicable interconnect schemes. Therefore NMOS design strategies may be said to be topologically "upward compatible" with technologies having an extra degree of freedom in their interconnect.

Regularity may be observed as a feature of good programming practice. Similar operations should be tackled by similar strategies. Routines should be written and used to perform a common operation on similar objects. These techniques reduce the complexity of a program by extracting the regularity in the problem being addressed. Regularity in VLSI design fulfills the same function.

3.1.4 LOCALITY

A module is a functional entity with a closely interconnected internal structure and a physical realisation within a well-defined boundary. The principle of locality restricts the member objects of a block to be accessible only inside that block such that the block

interacts with the outside world only through a clearly defined interface. In IC design a simple summary of this principle would be "put things into boxes". The interface is via structural connections which must exist at the boundary of the physical block. If a block were not physically inviolate then other primitives or modules could interfere with the internal artwork, and therefore structure, of a block in a context dependent way. This is operationally undesirable since it results in a multiplicity of cell types and violates the principles of hierarchical design. By way of analogy with software it can be argued that global variables detract from program structure by lending opportunities for the misinterpretation of the function of a variable by different code modules [Wulf73]. If the principle of locality is ignored in either discipline then the programmer or designer is forced into an unnecessary and wasteful acquisition of detailed global knowledge of the design which detracts from the effort applied to solving the local problem.

However, it may be observed that global variables can be used constructively in programming languages by employing them as parameters to the program, specifying their function clearly and disallowing their reassignment e.g. the constinteger in IMP [Robertson77]. This particular use of global variables may be compared with

the parameterisation of a structured IC design architecture at the highest level e.g. the width of a data word in a microprocessor.

By preserving locality a well structured design minimises global wiring. This increases design density since global wires consume an inordinate amount of chip area. Also, by placing structures in the lowest possible levels of the hierarchy and composing a design through repetitions of basic modules, a decrease in the total volume of design data may be achieved.

One of the most important benefits of the principle of locality is that checking functions must be applied only once, to the definition of a block, and not to every instance. The implication is, of course, that verification can thereby be made very efficient in comparison with fully expanded layout.

3.1.5 PROGRAMMABLE ARRAY STRUCTURES

Simple repetitions of user or system defined blocks can be extended to include structures such as read only memories (ROMs). In this type of structure the basic module has a number of alternative forms which have different internal descriptions but conform to the same interface e.g. a ROM array requires only two types of

cell, 0 and 1 storing.

A more complex example is given by the PLA structure c.f. Figure 1-5. It is derived from a set of modules, some of which have a number of alternative realisations. Fitting together selected modules from this set builds up the realisation of a logic function. Both ROMs and PLAs are regular implementations of "random" logic.

A software analogy to the programmable array structure is the use of data driven programs e.g. simple syntax analysers. According to the definition of the phrase structure, which must conform to the rules set down by the program, a syntax analyser will produce a variety of different results. Similarly a PLA framework can perform any of a set of logic functions, as determined by its size.

3.1.6 PARAMETERISED BLOCK DEFINITIONS

The full capabilities of the principle of block parameterisation can be arrived at through user defined modules. This allows stretchable cells via simple geometric translation facilities and more complex alternative realisations of the same function using common programming language control structures. These representations of design may be termed procedural models

and are a particularly powerful design technique.

It is also possible to generate different, though in most cases similar, functions which can be generated from the same root definition by including some conditional structures. Alternatives can be selected by parameters on the instance of the definition.

The existence of parameterised block definitions derives from the ability to define a VLSI design in a language with block and parameterisation features. In this case the relationship with software is not by analogy but by equality.

3.2 DESIGN PROCEDURES

Under the general guidelines for structured design it is possible to produce IC designs which exhibit the desired properties of regularity, hierarchy and systematic interconnect strategies. This section discusses design procedures through which this end result may be achieved [Gray79B]. These procedures are suggested from observations of structured design practice [Mead80, Masumoto78]. The main stages in the design process are

(i) architectural design, including floor-planning and hierarchical partitioning into blocks

(ii) block or cell design, possibly using stick diagramming techniques as an intermediate stage before artwork

(iii) chip integration i.e. assembling the chip according to the architectural design using the cell layouts.

Each of these stages requires graphical feedback to the designer and entails the use of various verification aids. A number of iterations through the design stages are to be expected before a satisfactory design is achieved. Following completion the design is sent for fabrication.

There are two views of the design activity which derive from to slightly different philosophies. Both are based on top-down design and bottom-up construction and the traditional procedure emphasises geometric refinement in descending the levels of abstraction in the architecture, with appropriate iteration taking place. The more novel approach is to use automatic rather than manual methods in the chip integration phase. This is not the standard cell library technique; the chip integration still follows the architectural floor-plan but the low level stretching and fitting together is done by program. Therefore, while parameterisation is useful in the original design process. it is absolutely essential in the algorithmic approach. To make use of the more powerful design aids the designer

must employ a higher degree of design structuring.

The system discussed in this thesis relates to the philosophy of structured design using language based descriptions and parameterisation. Two other systems viz. Bristle Blocks [Johannsen79] and the Chip Assembler [Tarolli79], also use this approach and are sometimes referred to as silicon compilers or silicon assemblers. The main goal of these systems is to provide a framework in which parameterised physical descriptions of layout can be described and manipulated with ease by the designer. PLA generators and automatic routers can be included as useful subsystems. Bristle Blocks also allows the designer to enter other representations of the design i.e. other than artwork, which may subsequently be used for documentation.

There are two major differences between Bristle Blocks or the Chip Assembler and ICSYS. The first is that in Bristle Blocks the cell parameters are assigned automatically by the system according to global considerations whereas in ICSYS the designer instances the cells with the appropriate parameters according to the needs of the architecture.

The second difference is the most crucial. Bristle Blocks and the Chip Assembler are physical design aids.

They make no provision for including structural or behavioural data in the design description and therefore, by the arguments of Chapter 2 do not provide comprehensive aids to verification. ICSYS regards the structural and behavioural design descriptions as important as the physical and is therefore able to provide functionally a more comprehensive service to the designer.

3.3 SUMMARY

Structured design is a methodology which has developed in order to control complexity. Structured programming is the application of these techniques to software and has been investigated in depth over a number of years. Structured design of ICs, the hardware equivalent, is a less mature field. However, it has been shown that many of the principles of structured programming are analagous to the principles of structured IC design. This is an exciting observation since it implies the possibility of valuable cross-fertilisation of ideas between the two areas of interest.

4. MODELS FOR IC DESIGN

A structured design style is a practical and attractive method of controlling the complexity of VLSI designs. For a design system to aid a designer it too must work within the principles of structured design and must deal with the problems of both design description and design verification. The following three chapters describe the original contribution of the author to this area of research.

Models are used to build design descriptions that may be algorithmically tested. Different data is required for each process and this data is usually extracted from two of the three fundamental descriptions of the design. For example, simulation uses structural and behavioural data together with test stimulii to produce circuit responses. Also, for non-pathological dimensional design rule checking it is necessary to be aware of both the physical and structural aspects of a design. Thus complete verification requires that the structural, physical and behavioural attributes of the design be known to the design system. In addition, these descriptions must be complete and consistent at every hierarchical level of the design from primitive components up to high level blocks to achieve reliable verification. Higher level behavioural descriptions can be captured using procedural

models in the normal way. The most important need, therefore, is for models to describe a design that allow the capture of structural and physical data. Proposals for a sufficient set of models are given in the next section. These are based on notions of capturing design intent by using initial topological specifications.

It is important to recognise that the use of such models is a departure from traditional methods of IC design which concentrate on the geometric description of the artwork. The coordinode is the key new object which unifies the design description over the three domains. The primitive components i.e. wires, transistors, contacts and pins, reference these new objects in constructing the description of the design. Although this approach bears some similarity to symbolic layout it differs in that there is no layout penalty caused by excluding human layout skills, as is possible with systems that restrict layout to a grid [Gibson76] or automatically produce artwork from a stick diagram [Williams77]. Top down design and the use of hierarchical blocks, or cells, is not a new idea in IC design. However, the commonly available simple duplication of geometry is too limited a facility. A more powerful mechanism is necessary to control the complexity of VLSI design. This is implemented through parameterisation of the block description. From within the environment of an object

oriented, general purpose programming language the designer is free to use the full set of control mechanisms including scaling factors, loops and conditionals.

4.1 THE COORDINODE

The coordinode is a named object which has structural, physical and behavioural significance in the IC design process.

Structurally its function is to join together components i.e. wires, transistors and block instances. From a topological standpoint, a graph model could be constructed in which coordinodes are the nodes while components are the branches. A set of components which reference the same coordinode are connected by it. Figure 4-1(a) shows the layout for a block containing half a shift register bit i.e. an inverter with its output controlled by a pass transistor. An associated stick diagram, annotated with coordinode names, appears in Figure 4-1(b).

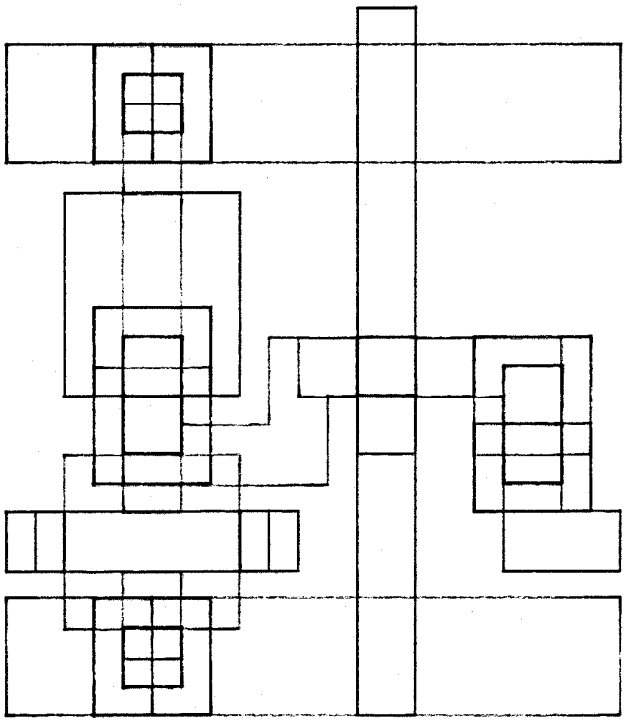


FIGURE 4-1(A). SHIFT REGISTER CELL - LAYOUT.

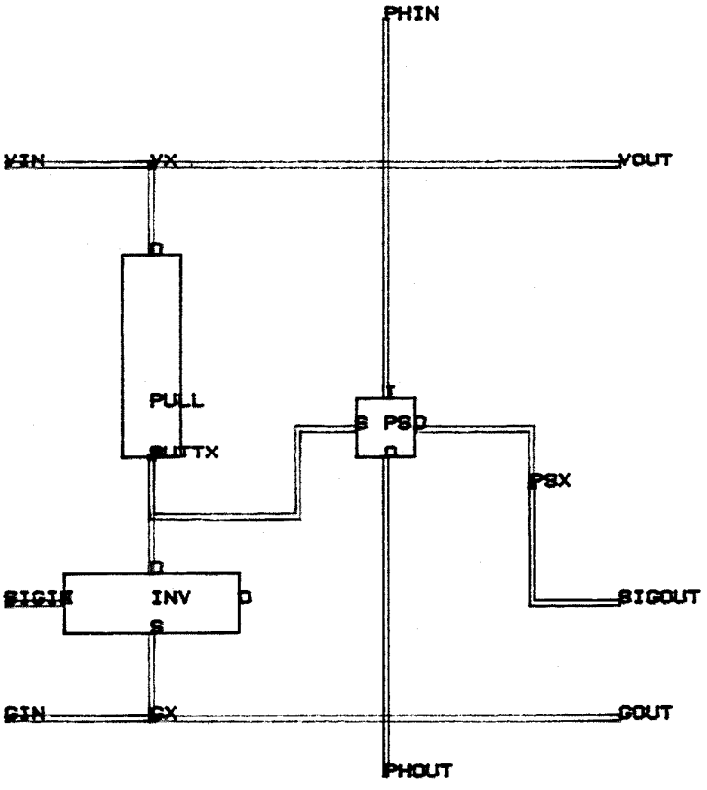


FIGURE 4-1(B). SHIFT REGISTER CELL - STICK DIAGRAM.

In addition, a coordinode may be designated a "pin" i.e. an interface point to an external environment, or a "contact" i.e. a reference to components on more than one layer that the designer intends to be joined. The absence of a contact under such circumstances is held to be an error since design intent must be positively reinforced. More than one coordinode may exist at the same physical position while having different structural characteristics.

Physically, the coordinode is placed in the 2D plane. This fixes the position and extent of primitive components since they are sited relative to coordinodes. A coordinode which acts as a contact has a physical realisation which is the appropriate geometry for the type of contact. The designer may define the orientation of a butting contact. Pins have no geometric significance.

Behaviourally, the coordinode belongs to an electrical node or net which carries a simulation state between the components it joins.

To summarise, the attributes and functions of a coordinode are:

(i) To provide connections between components.

(ii) To provide inter-layer connections by the

contact mechanism.

(iii) To provide block instance connections by the pin mechanism.

(iv) To place components within blocks and define their sizes.

(v) To pass simulation states between components, block instances and procedural definitions of block behaviour.

4.2 PRIMITIVE COMPONENTS

Traditional IC design concerns itself with only the geometric, usually polygonal, description of the circuit. While this representation is adequate for fabrication it is not suitable for a generalised structural description and associated verification procedures. In any CAD system it is necessary to define a number of primitive components for modelling purposes. The art is to choose models that allow structural, physical and behavioural design intent to be captured concurrently while maintaining consistency of the design over these representations at all times.

4.2.1. THE WIRE

The simplest component is a wire. It exists in a single layer and connects two coordinode endpoints by a path whose width defaults to a minimum associated with the

layer (Figure 4-2).

Structurally, or topologically, the wire is a single branch in the circuit graph. Connected wires form a net i.e. an electrical node in the circuit.

There are two alternatives for the geometric realisation of a wire. The difference lies in the way that coordinode endpoints are dealt with when the path describing the route that the wire is to follow is made into a physical track of a certain width. The wire may be curtailed at the endpoint i.e. the endpoint lies on the perimeter of the track (Figure 4-3(a)), or the inflation process which produces the track from the path may be extended to include the endpoints (Figure 4-3(b)).

At first sight the second alternative provides the most attractive model. Two wires which meet at right angles may not form a correct, i.e. design rule satisfying, connection under the curtailed option (Figure 4-4(a)). This problem does not occur with the full inflation option (Figure 4-4(b)).

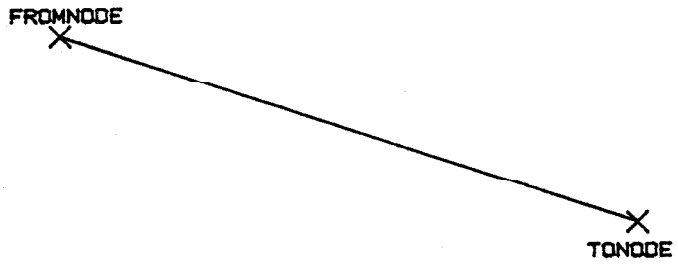


FIGURE 4-2. WIRE MODEL - STRUCTURAL DESCRIPTION.

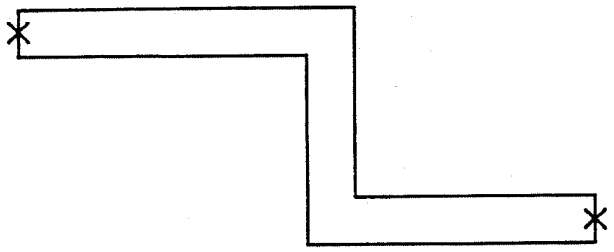


FIGURE 4-3(A). WIRE MODEL - CURTAILED ENDPOINTS.

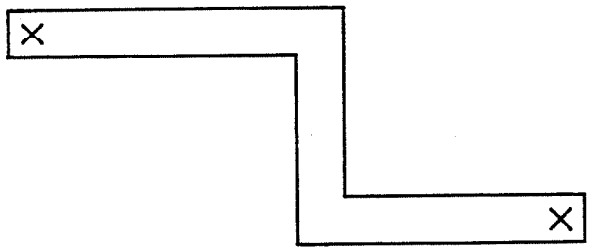


FIGURE 4-3(B). WIRE MODEL - INFLATED ENDPOINTS.

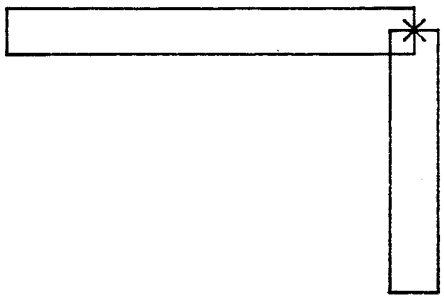


FIGURE 4-4(A). WIRE CONNECTION - CURTAILED.

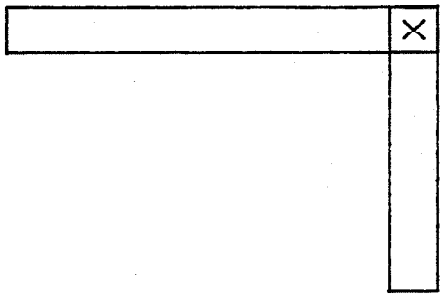


FIGURE 4-4(B). WIRE CONNECTION - INFLATED.

However, this type of connection does not seem to occur in practice, at least within the context of designs described using the models proposed in this thesis. This problem was not encountered in the examples given in Section 6.2. In addition a number of undesirable effects become evident during circuit construction. Consider a T-piece connection of wires on the same layer with another wire above and parallel to the cross piece and separated by the minimum distance defined by the design rules. The vertical wire is wider than the cross piece. The curtailed option yields a correct construction (Figure 4-5(a)) but the full inflation option contains a design rule violation (Figure 4-5(b)). For clarity the two half cross pieces, which also join at the connect point, are shown as merged.

Another example is provided by the butting contact. The physical realisation of the butting contact is governed by its model (Figure 4-6(a)). Polysilicon and diffusion wires connect to it at the coordinode which lies at the midpoint of the contact. Under the curtailed option a correct construction is obtained (Figure 4-6(b)) but under the full inflation option the polysilicon track extends too far under the contact hole and introduces a design rule violation (Figure 4-6(c)). These reasons make the curtailed track option a more attractive model.

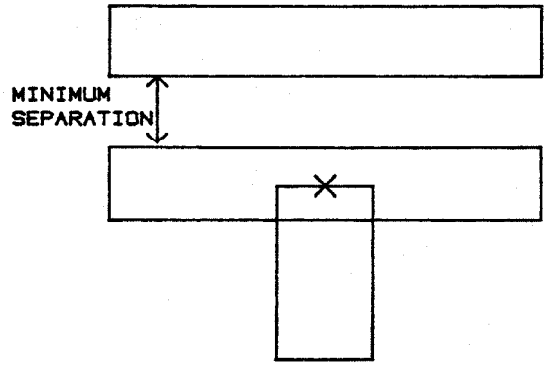


FIGURE 4-5(A). T CONNECTION - CURTAILED.

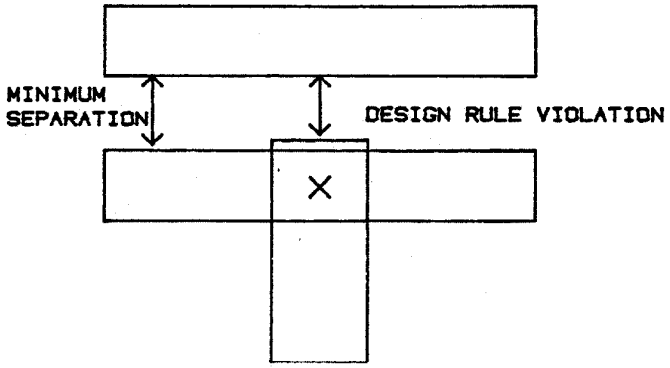


FIGURE 4-5(B). T CONNECTION - INFLATED.

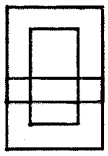


FIGURE 4-6(A). BUTTING CONTACT.

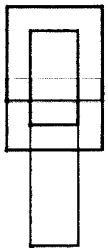


FIGURE 4-6(B). BUTTING CONTACT WITH CURTAILED WIRE.

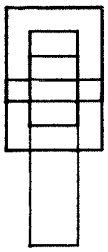


FIGURE 4-6(C). BUTTING CONTACT WITH INFLATED WIRE.

The problem of non-connecting wires (Figure 4-3(a)) is then solved by generalising the model for contacts. Contacts, butting or otherwise, possess physical models e.g. Figure 4-5(a). The description of the IC design makes it possible to discover whether or not two wires referencing the same coordinode exist in the same layer. If they do then the design environment's physical model of this connection causes a "contact" box to be placed on that coordinode. Its width is the least width of the wires connecting to that node. (Figure 4-7).

Behaviourally, a wire is part of an electrical node which may connect several components.

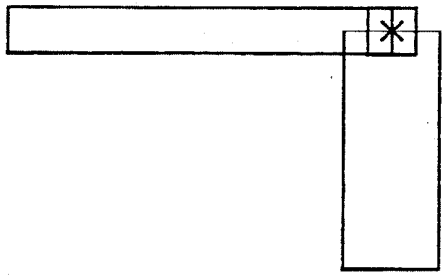


FIGURE 4-7. WIRE CONNECTION WITH CONTACT.

4.2.2 THE TRANSISTOR

The transistor is the key primitive functional element (Figure 4-8). Different types of transistor may be modelled in the system e.g. pullups, pulldowns and pass transistors. Transistors are given textual names by the designer.

Structurally, the transistor has gate (input and output), source and drain connections.

Physically, the transistor possesses a path either between its gate input and output or its source and drain. The default path is determined by the type of transistor. This defines the electrical parameters of the device and its geometry. Note that the physical realisation of the transistor extends outside the overlap area to include the abutting design rule enforced regions on the polysilicon and diffusion layers. Additionally, depletion mode transistors have an associated implant geometry.

Behaviourally, the different types of transistor act as specified by their associated models and according to the simulation states present at the coordinode connection points.

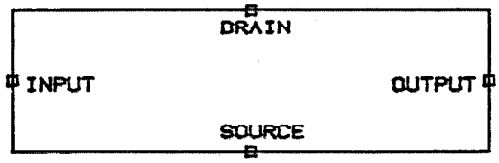


FIGURE 4-8(A). TRANSISTOR - STRUCTURAL MODEL.

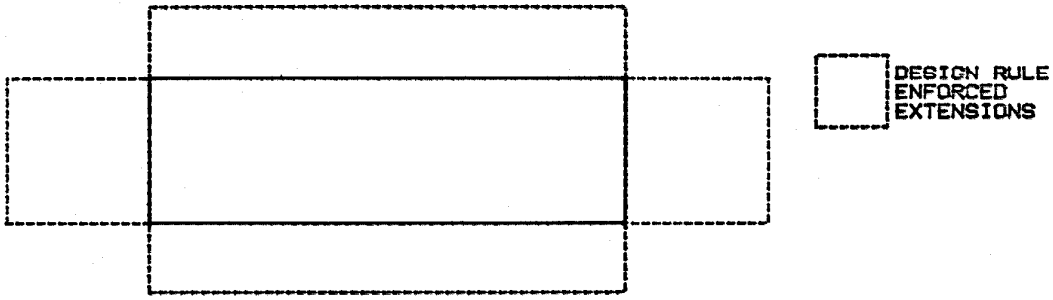


FIGURE 4-8(B). TRANSISTOR - PHYSICAL MODEL.

4.3 BLOCK STRUCTURE

A structured IC design methodology derives its power from the control of the circuit complexity that may be gained by partitioning the circuit into hierarchical blocks.

4.3.1 THE BLOCK DEFINITION

Block definitions are discrete modules with structural, physical and behavioural descriptions. Structurally they consist of a set of components referencing a set of coordinodes. The components may be primitives or instances of other blocks.

Physically a block is a bounded region on the surface of the silicon chip. Both bounding boxes and bounding polygons may be used in the physical description but the box alternative is more efficient for cell arrays, chip assembly and computation while not being unduly restrictive within a structured design methodology. Behaviourally, a block performs as the aggregate of its parts, although when instanced it may have an associated procedural model that has been provided by the user.

The regularity of the design i.e. the use of arrays of cells and the control of interconnect, further contributes

to the reduction in the volume of data. With this goal in mind a single block may be instanced many times. A block instance may be referenced within another block definition so that a hierarchical structure with a number of levels of block instancing can be built.

4.3.2 BLOCK PARAMETERISATION

A block definition may be parameterised to achieve a number of effects which are important in chip assembly [Johannsen79].

(i) deformation or stretchability
(ii) variable dimensions in arrays
(iii) programmability of function e.g. ROM, PLA
(iv) conditional inclusion of circuit elements
(v) sizing of circuit elements

Thus parameterisation is an extremely powerful tool in the construction of a design enabling large savings of effort in designing separate cells for similar functions. For example, deformations and iterations of a half shift register cell as shown in Figures 4-9 and 4-10 respectively.

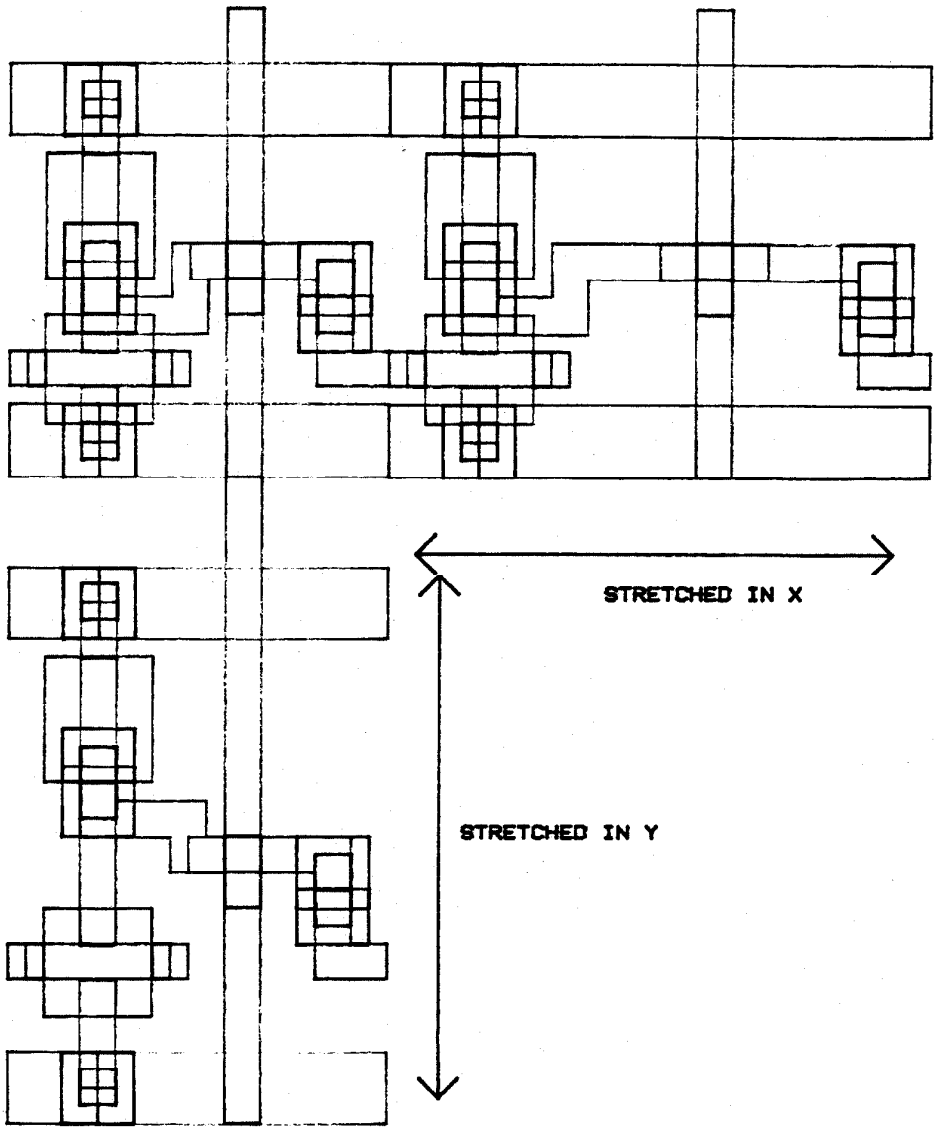


FIGURE 4-9. SHIFT REGISTER CELL - DEFORMATIONS.

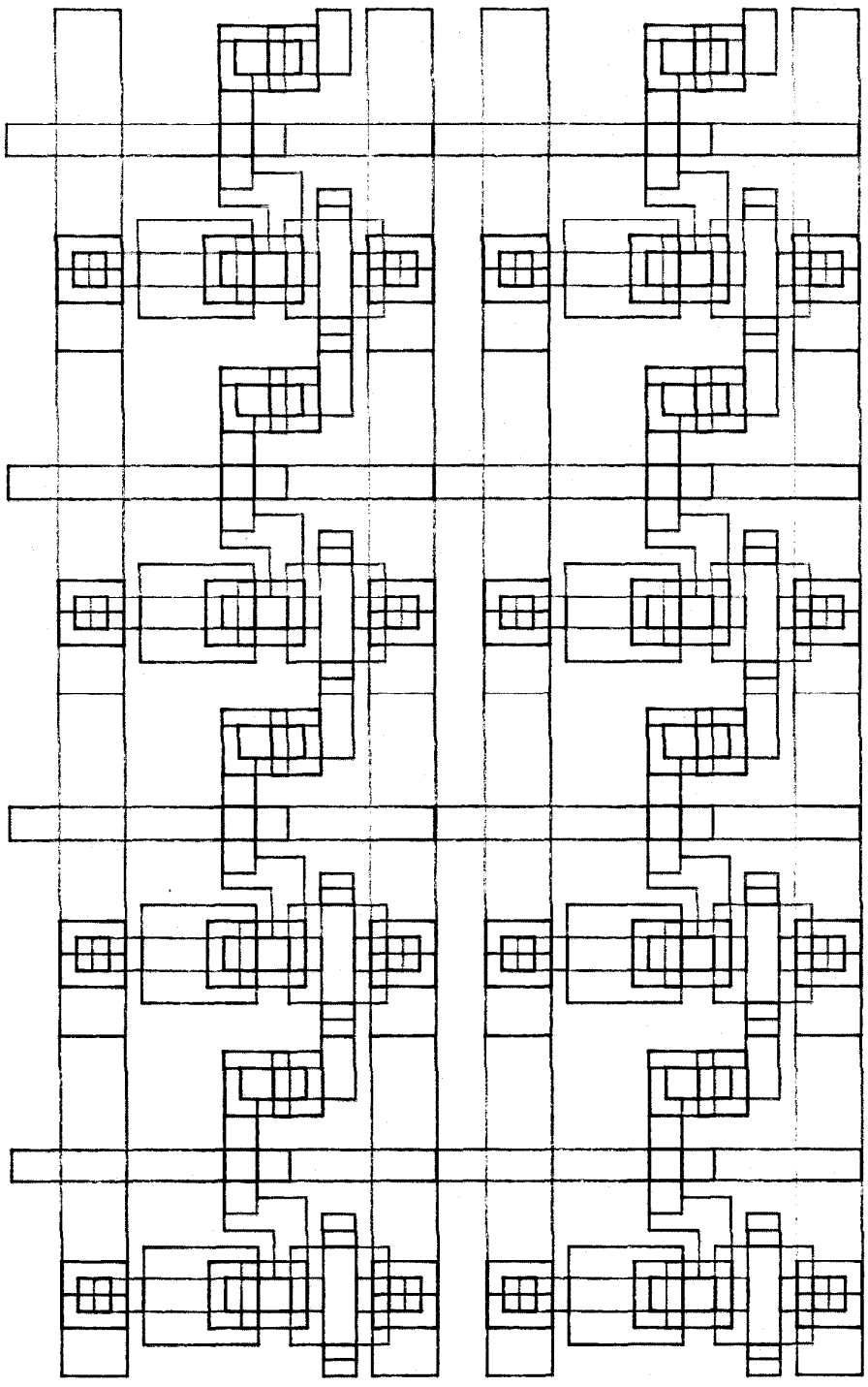


FIGURE 4-10. SHIFT REGISTER CELL - ITERATIONS.

4.3.3 THE BLOCK INSTANCE

The block instance is a named object which may be viewed as a user defined component. It references its parent definition and is parameterised by a transformation matrix.

Structurally, a block instance is a set of pins interfacing to other components contained in its enclosing block definition. Thus pins specified in the block definition are translated into coordinodes in the calling environment within which they have the appropriate structural significance.

Physically, a block instance is fully expanded into its components for fabrication but may be represented as an outline on check plots and in dimensional design rule checking.

The behavioural characteristics of a block instance may be inferred from its parent definition either by expansion of the contents of the definition or by execution of a simulation procedure. In either case the block instance must preserve its local state as distinct from the characteristics it shares with other instances of the same block definition. If this were not the case then two instances being simulated concurrently would interfere

with each other.

4.4 SUMMARY

A set of models have been defined which describe primitive components and block structures in IC design. These models possess structural, physical and behavioural attributes and can be used to describe a circuit implemented in silicon. The implementation of these models within a general purpose programming language, SIMULA, will be described in Chapter 5.

5. AN IC DESIGN ENVIRONMENT

Matching the design system to the philosophy of the design methodology provides the designer with a rich environment in which to explore design ideas. It is a development in design aids which holds great promise of gains in functionality and performance. The purpose of this chapter is to describe the characteristics of an IC design environment which embodies the methodology of structured design and provides facilities for design description and verification.

The first section explores the subject of textual descriptions of an IC design in general terms without reference to a specific language. Encasing design related constructs within an already existing programming language is recognised as a particularly powerful scheme that has a number of advantages. The second section concentrates on the methodology of design within the proposed environment.

The remainder of the chapter is devoted to the description of an IC design system written in SIMULA and based on the ideas contained in this thesis. The topics of design description and design verification are dealt with in two separate sections.

The design verification subsystems differ from usual

practice in that they draw their data from an integrated description of the IC design which is consistent over the structural, physical and behavioural domains. It is this factor that contributes to their reliablity and completeness. In general the subsystems themselves follow common methods of operation and are therefore described only briefly. Novel or unfamiliar procedures are treated in greater depth.

5.1 LANGUAGE DESCRIPTIONS

This section explores a number of interesting issues concerning the textual description of an IC design. Relating structured design to structured programming is naturally one of the most important. However, languages differ in their philosophy and it is contended that the implementation of an IC design environment must be based on a particularly highly structured and powerful general purpose programming language. Those features of a programming language which are essential to IC design are singled out for closer examination. Finally, consideration is given to the advantages of general purpose over special purpose languages for design.

The analogies between structured design applied to VLSI systems and the structured design of software systems have already been discussed in Chapter 3. The recognition of a

close relationship between the principles of hardware and software design is, of itself, a conceptually interesting subject for study. Tangible rewards can be obtained by applying the lessons of one discipline to the other. By using software in the design process these links become even more obvious, the distinctions between the fields of application blur, and the attractions of using a block structured language, the major tool of structured programming, are strikingly apparent.

The general benefits of block structured languages have been well recognised for some time. Their suitability for IC design stems from the procedure construct, its capacity for parameterisation and the variety of possible control structures. However, within the set of block structured languages the various members have different capabilities. Of particular interest in design software are object oriented languages. These have a number of advantages over languages which lack that facility. They provide greater rewards through the added structural organisation and descriptive power that they contain. In particular a programmer can associate with an object, or module, not only data attributes, as is ordinarily possible in a record oriented language, but also procedural attributes. This serves to further localise all the information concerning an object and promotes correct implementation. Most block structured languages allocate storage space

from a stack. This restricts the creation and deletion of data records unnecessarily and often results in wasteful static allocation of large arrays which must then be handled by the system implementer. Object oriented languages employ a heap to overcome that problem and so allow the implementer to concentrate on the higher levels of design software.

Programming languages have a number of features which can be usefully employed in hardware design. Using a general purpose language as a design medium provides a very rich environment in which a designer can take full advantage of the available data descriptive mechanisms and control structures e.g. conditionals and loops. The procedure is a convenient higher level view of a design module and the differences between similar blocks can be dealt with by program description as opposed to a proliferation of separate data files.

In this view of design description parameterisation of classes or procedures is of fundamental importance. Parameterisation of objects provides the appropriate mechanism for the tailoring of specific design modules from a generalised description. Thus again the strengths of the programming language are translated into powerful design aids.

A further aspect of programmatic IC descriptions has still to be considered; the issue of general purpose versus special purpose languages. A general purpose language based design environment has great flexibility of form within which individual designers can pursue different design themes, perhaps in a direction the creator of the basic design environment did not originally foresee. Flexibility and extensibility are therefore built-in advantages of such a design system. Fast implementation and reimplementation are additional virtues of general purpose language based systems. Technology is forcing the pace of IC design aid systems and the ability to write useful and relevant software quickly is vital. Special purpose languages are difficult to define, take a large amount of effort to implement and are expensive to maintain and extend.

5.2 IMPLEMENTATION STRATEGY

The IC design environment provides the designer with a set of procedures which mirror the models of primitives and block structure identified in Chapter 4 as being necessary within the design system. Primitives are referenced by procedure call. The block structure and parameterisation of the design maps directly onto the block structure and parameterisation provided by the language.

The creation of a module occurs when the code describing that module is executed. This causes an object to be built which contains all the relevant structural, physical and behavioural information. This data is kept as the data and procedural attributes of the object.

The system provides documentation and verification services for the designer via system procedures which interpret and perform calculations on the data structure. Therefore each object, whether block or primitive component, has a procedural attribute corresponding to each of the facilities which is relevant to that object. A menu gives the designer access to a list of options. Some merely set up parameters; others cause the execution of procedures which follow the hierarchical design through the block instance mechanism to the primitive components using the recursive descent technique.

5.3 SIMULA IMPLEMENTATION - DESIGN

The programming language used in this exercise is SIMULA, an object oriented language based on ALGOL 60 [Birtwistle73]. Experience with other languages e.g. IMP [Stephens74] and FORTRAN, which are not object oriented has shown that their structural and descriptive inadequacies are a severe hindrance in this field of application. The major problem, as has already been

indicated, is the lack of freedom in dynamic storage allocation. The stack has been found to be too restrictive in this type of application. The process of design requires that descriptive records be frequently created, edited and deleted. Therefore, when an object is created during the execution of the program a storage area is dynamically allocated for the object from the heap. The storage area contains items of data and references to other objects.

There were a number of reasons for the selection of SIMULA as the implementation language. The fact that it was the only object oriented language available on any machine to which access could be obtained was the major factor. Other possible languages have neither its richness of structure nor its sophistication of run-time storage allocation. The lack of a heap is a particularly important drawback as it commonly involves the systems implementer in management of memory at a low level i.e. the allocation and deallocation of differently sized memory areas and associated garbage collection. These are tedious and non-trivial facilities to construct, often machine dependent, difficult to make efficient and uncomfortable in their interface to the rest of the applications system.

SIMULA is not ideal. Being based on ALGOL means that

it suffers from deficiences in its forms of control structures e.g. loops, and its input/output facilities are poor. Also large programs which should be split up over a number of files are forced into a linearly dependent CLASS structure which does not always fit the organisation of the program. However, although several interesting languages have been reported in the literature e.g. MESA [Mitchell78] and ADA [Ichbiah79], none of these are generally available and the task of designing and implementing one of these or an equivalent was too onerous to fall within the scope of this project. As the work has progressed SIMULA has emerged as a good choice in that it has enabled software to be exchanged with other researchers, principally those at the California Institute of Technology. The most useful modules have been the graphics subsystem [Wipfli78] and the polygon package [Darton80]. Unfortunately, the DEC-10 version of the compiler and run-time system have not been able to cope as well as had been hoped with the resource requirements of the design system in terms of processor time and, more crucially, storage space. The consequence of this problem is that some of the examples of verification procedures are illustrated by smaller and less complex designs than would be ideal. However, it is believed that the general philosophy of the system has not been compromised by this failure and this issue among others is discussed in Chapter 7.

An IC design may be described within a special purpose SIMULA programming environment. This environment provides a block structuring mechanism within which cell definitions may be constructed and includes procedures for generating and connecting components. The normal CLASS parameterisation can be applied to block definitions with great effectiveness for cell deformation and the SIMULA sequence and control structures can be used for iterations and conditional composition.

The following sections deal with the detailed implementation of the primitive component models and the block constructs c.f. Chapter 4. A complete example of a shift register cell array is contained in Appendix A.

5.3.1 THE BLOCK DEFINITION HIERARCHY

The special purpose programming environment provided by the system includes a CLASS "blockdef". Any user defined block definition is constructed as a subclass of blockdef e.g.

```
blockdef CLASS shiftregistercell;
```

This implies that the procedural attributes of blockdef are also in its subclasses. Specifically, procedures exist within "blockdef" which allow the designer to create primitive components and instances of other blocks within the new block definition.

Parameters of a block definition appear in the CLASS header e.g.

```
blockdef CLASS cell(gnd,vdd,signal);
REAL gnd,vdd,signal; VALUE gnd,vdd,signal;
```

The designer includes a block instance in a block definition by calling a procedure which takes as parameters the name of the instance, a reference to a block definition object and a transformation matrix which physically positions the instance. Thus a "shiftregistercell" object may be created e.g.

```
src:-NEW shiftregistercell;
```

Within another block definition a procedure call may appear which references it, positioning the instance at the origin of the new definition. The orientation of the block instance is left unchanged by including the identity matrix as its transformation matrix e.g.

```
blockinst("shiftcell",src,NEW transform(NONE));
```

Transformation matrices may be constructed to perform translation, rotation and reflection using procedures included in the CLASS transform [Wipfli78].

A block instance is regarded as a user defined component. Structurally it is viewed as a black box with a number of pins to its calling environment. Each pin is translated to a coordinode in the calling environment and is referenced by concatenating the instance name and pin name e.g.

"shiftcell.signalinput"

These names are used by the new block definition in attaching other components to the block instance.

Physically, the block instance is positioned by the transformation matrix and it follows that the placement of the coordinode in the calling environment is fixed by the same matrix. The block instance covers the same dimensions in physical area as its parent definition.

Behaviourally, the block instance may either act as the aggregate of the components contained in its parent definition or, if the designer so chooses, perform according to a procedure included in the block definition.

Thus through successive block instancing at a number of levels a hierarchy of modules is constructed and the volume and complexity of design data is controlled.

5.3.2 PRIMITIVE COMPONENTS

There are two basic primitive components viz. the wire and the transistor. Structurally, wires connect two coordinodes and exist on a single layer e.g.

```
wire("from","to",poly);
```

Physically, a wire is a path joining two points in the 2D plane. The default path is a straight line. Any other

path is constructed by system provided procedures which describe the increments for each step of the path [Locanthi78]. Both absolute and relative increments are permitted and steps may be paraxial or generalised vectors e.g.

```
wire("from","to",poly).path.x(10).dy(1).xy(12,6);
```

This statement describes a wire on the polysilicon layer which connects coordinodes "from" and "to" with a path which starts at the position of "from", say (fx,fy), and then proceeds incrementally to the points (10,fy), (10,fy+1) and (12,6). The path is automatically attached to "to" if the last increment does not complete the full path.

The specification of L-shaped jogs may be abbreviated using the procedures "xtheny" and "ythenx" e.g.

```
wire ("from","to",poly).path.xtheny;
```

This is a very useful and concise way of describing a path since it allows the specification of a connection by relative position and without recourse to numeric values of displacement. Width is a default associated with the layer unless changed explicitly by the designer e.g.

```
wire("from","to",poly).width(10);
```

Behaviourally, a wire is one component in an electrical node or net.

There are a number of different types of transistors.

Pullups, pulldown and pass transistors are currently defined in the system. The models for each type differ in some respects but are basically the same.

Structurally, the transistor is a black box with four connections. These connections are pre-named "src","drn","in" and "out". A procedure is provided by the system which includes a particular type of transistor within a block definition and allows the designer to name it e.g.

```
pulldown("inv");
```

This statement causes coordinodes, named by concatenating the transistor name and connector names, to be entered into the block definition where they may be referenced by other components e.g.

```
"inv.src"
```

Physically, for patterning purposes, the transistor is a diffusion and polysilicon overlap area, whose dimensions are determined by the placement of its coordinode connections. Design rules enforce abutting extension regions of particular widths in the polysilicon and diffusion layers. Pullups are plotted and fabricated with an implant layer. Pulldowns are assumed to need a gate extension region, which by default continues in the same direction as the gate path, but may be specified by the designer. The gate path itself follows the same defaults

as wire paths with the same arrangement for defaults.

Behaviourally, each transistor acts according to its particular system defined model. A pullup is always conducting since it has its gate tied to its source. A pulldown or pass transistor is conducting if a logic 1 is applied to its gate.

5.3.3 COORDINODE ATTRIBUTES AND PLACEMENT

Coordinodes may be explicitly assigned one of two attributes, pin and contact e.g.

```
pin("sigin"); contact("gndx");
```

A pin is simply a structural attribute which defines a block's interface to its calling environment.

A contact acts as confirmation by the designer that components on different layers which reference the same coordinode are logically connected. Physically, contacts require a contact hole and a design rule enforced overlap on the connecting layers. Butting contacts are placed by the system if a polysilicon to diffusion connection is required. The orientation of the contact may be deduced from the directions of its connections in the direction of the diffusion overlap but it may need to be explicitly defined by a vector e.g.

```
contact("buttx").orientation (1,0);
```

Coordinodes are placed at a point in the plane, which may be a function of the block definition parameters. For example, baselines are positions on the x or y axis which may be parameters to the block definition and may be used to produce stretchable cells [Johannsen78]. Coordinodes may also be placed relative to another's position e.g.

```
place("sigin",NEW point(0,sig));
place("inf.src",NEW point(inv+3,sig+4));
place("inv.drn",position("inv.src").plus (NEW
    point(0,2)));
```

5.4 SIMULA IMPLEMENTATION - SUBSYSTEMS

A number of subsystems are provided within the IC design environment. These provide graphical feedback, verification and access to fabrication. Chapter 6 includes examples of designs processed by the system. This section describes the facilities available and addresses the techniques by which they have been realised. A list of the menu options available to the designer at the highest command level are shown in Figure 5-1. Some subsystems include local menus. The system takes advantage of the hierarchical nature of the design and uses recursive descent of the block instance tree in most of the subsystems.

```
        help
annotate - set switch to display names
block - choose a block definition
build - construct a block definition
cif - produce a cif file
connect - check connectivity violations
device - switch graphical output device
drc - dimensional design rule check
electrics - electrical design rule checks
grid - draw grid on graphic device
help - print this information
layers - select layers to plot
level - depth of nesting plotted
list - list block definition dictionary
logic - logic simulator
names - annotate with selected names
plot - plot physical layout
spice - spice node & component list
sticks - stick diagram
stop - return to monitor level
window - delimit plotting area
diag - switch diagnostics off & on
command:
```

FIGURE 5-1. MAIN COMMAND MENU.

5.4.1 GRAPHICAL FEEDBACK

An IC design may be viewed at a number of levels of abstraction. This design system supplies both the usual form of artwork and a stick diagram representation as feedback to the designer. Design modules may be represented by their outlines and the level of the hierarchy to which the design is plotted is under the control of the designer. Graphical output may be annotated with block, transistor and coordinode names and a grid is optional. The graphics package is essentially device independent and interfaces to another module containing drivers for several devices [Wipfli78]. It has been tailored to a recursive programming environment and contains a transformation stack.

The algorithms to display or plot either artwork or stick diagrams are essentially the same; the main difference is that in a stick diagram wires are plotted as standard width narrow tracks. The procedure is as follows:

(i) The data structure is traversed once for each layer plotted.

(ii) The current block definition's component directory is processed and primitive components are output directly.

(iii) Block instance components cause their particular instance transformations to be concatenated with that of the calling definition i.e. the transformation in current use. The algorithm then recurses on the parent definition of the block instance.

The level of the hierarchy to which the algorithm descends is under the control of the designer. Options are also available to plot a grid and to plot layers of the design selectively.

In general, the artwork included as figures in this thesis does not show the implant layer. This is an option within the system and is employed in this context to make the artwork easier to understand by reducing the number of lines and avoiding the use of one colour for two different masks.

Stick diagrams are a very useful representation of an IC design; the structure of the design is made evident while at the same time the topology, layer assignments and device sizes are still available for inspection.

Coordinode and device names can annotate either artwork or a stick diagram though they are generally more useful with the latter. Unfortunately, close placement of

devices and coordinodes with long names can result in overstriking to the detriment of the legibility of the annotation. This effect increases as the scale of the plot decreases e.g. the A4 size appearing in this thesis. This is a cosmetic problem and could be solved in a production system.

5.4.1.1 GRAPHICAL DISPLAY

There seems to be a reasonable concensus of opinion that an interactive display device for use in IC design should provide both colour and filled-in shapes. A colour raster-scan display fulfils both of these objectives. This IC design system interfaces to such a display. It was designed by C. Minter [Minter79] and a copy was built at Edinburgh University during the summer of 1979. A detailed description of the device will not be presented but its salient features are filled-in boxes and polygons, a 16 location colour look-up table (to select from a range of about 4000) and a resolution of approximately 512 by 512 pixels. This is believed to be a minimum level of functionality necessary for an IC design terminal. An increase in resolution would be desirable but an improvement in this area would require a higher quality monitor and a larger frame store.

5.4.1.2 PLOTTING

Hardcopy output is an obvious necessity to enable a designer to examine at length part or all of a design that is too large to be displayed on the colour graphics terminal. As with the display, colour and filled-in shapes are both very desirable. Unfortunately, as yet both of these features are offered together only on very expensive equipment e.g. a colour ink-jet plotter. Therefore the main alternatives are multi-colour pen plotters and electrostatic monochrome plotters, which each provide one of the desirable features. This IC design system is interfaced to the HP 7221A 4-pen flat bed plotter. It has a small plotting area of approximately A3 size. It is a relatively inexpensive device which is suited to low complexity output and low throughput.

5.4.2 MASK MAKING

All IC design systems must provide a method of interfacing to mask making equipment. Some systems produce pattern generator drive tapes directly [Eades76]. This system chooses to take a higher level route via a device independent intermediate form of textual design description, CIF 2.0 [Sproull80]. This output form is a simple graphics language which can be written and read with ease, thus saving effort in the design software and

allowing access to cell libraries as well as to pattern generators.

The algorithm for producing CIF 2.0 follows a similar path to the graphical feedback algorithm of Section 5.4.1 but does not need to descend the hierarchy of description.

(i) For each block definition in existence, and in the order in which it was created, it produces a CIF 2.0 description of the components it contains i.e. box, wire, polygon and symbol call statements.

(ii) Each component causes one or more CIF 2.0 statements to be output and may cause the current layer assignment to be updated.

The CIF 2.0 symbol call statement includes an instance transformation expressed as a series of translations, reflections and rotations. For the purposes of this and other design aids, the matrix representation of the transformation is preferable and is the form used internally by the system.

5.4.3 VERIFICATION

The term verification covers a number of different procedures that a designer may wish to employ to ensure

the correctness of a design. These will be dealt with separately in the following sections. Dimensional design rule checking and the calculation of electrical parameters will be tackled first and the subject of simulation will be discussed in the last section under a number of sub-headings.

5.4.3.1 DIMENSIONAL DESIGN RULE CHECKING

The object of this type of verification is to look for violations of the design rules e.g. line separations less than a minimum defined for the layer (c.f. Section 2.2.1). Design rules are compiled for a given process, expressed in geometric terms and distributed to designers who must comply with those rules to achieve a working, high yield circuit. Since this system subscribes to the philosophy of correctness by construction it is not necessary to check line widths, overlaps around contact holes or polysilicon and diffusion extensions at transistors. Checking a design rule involves extracting the relevant portions of geometry from the layout, applying some geometric transformations and testing for an error condition. For example, to test minimum separation of polysilicon wires, extract the polygons defining the boundaries of the wires, inflate each of the objects by half the minimum separation and check for overlap. Any overlap regions result from too close packing. These last

two operations are performed by a polygon package [Sutherland78, Barton80]. Examples of algorithms for specific design rule checks are included in Section 6.3.3.1. It is envisaged that most design rule checking will take place at the lowest level of the design hierarchy since design rule checking between instances of the same or different blocks does not require an enumeration of the instances or primitive shapes in each block. An abstraction of each block based on the shapes that are placed near its perimeter is sufficient to check design rules between blocks [Buchanan78].

5.4.3.2 ELECTRICAL CONSIDERATIONS

There are a number of electrical parameters which a designer might wish to compute e.g. the resistance of a wire, the ratio of a pullup to a pulldown, or the capacitive load on an driver. This system can perform such calculations since it models both the structural and the physical aspects of the circuit and, through the naming of components and coordinodes it can provide the designer with a familiar user interface.

For example, a transistor can be identified by name and a wire is specified by the names of its two coordinode endpoints. Both components and coordinode names are held in directories which may be searched to find the named

object. Calculations then proceed on the data already held in the internal data structure.

5.4.3.3 CONNECTIVITY

While the design construction model ensures that structural connections have correct geometric counterparts, unintentional physical connections may have occurred in the design and must be detected. This is accomplished using geometric analysis methods similar to those employed in dimensional design rule checking. From each separate layer, geometry is extracted according to its membership of an electrical node and any intersections between geometries from different nodes are detected as errors c.f. Section 6.3.3.3. Wires which are connected are part of the same electrical node. This is a clear example of a verification procedure depending on the system being aware of both the structural and physical descriptions of the design.

5.4.3.4 SIMULATION

Simulation of design behaviour may be performed at a number of levels of abstraction viz. circuit, logic and functional block. This is achieved either by integrating the verification subsystem within the design environment or by providing a facility for outputting data files to

drive independently operating simulators. The first option adds a significant overhead to the size and complexity of the design system but permits the simulator to be directed from within the same program as the other subsystems. The second option removes the simulation function from the design system proper by sending it to an external simulator through a data file interface.

For the simulators described in this section the choice of an internal or external procedure was based on suitability and availablity. Circuit simulation was regarded as unsuitable for inclusion within the system because of its high computational requirements and because of its availablity as a package. Logic simulation was integrated within the system because its level of operation suited the models of the design and it was possible to allow the designer to interact with the simulation. Block level simulation is implemented by user-written procedures and is therefore part of the design description within the system.

5.4.3.4.1 CIRCUIT LEVEL

Circuit level simulation is a complex problem which a number of people have tackled e.g. SPICE [Nagel75]. It is computationally expensive and used most frequently at the lowest level of the design abstraction i.e. with groups of

primitive components. Given these characteristics the most reasonable course of action for the design system was to provide a subsystem which produced a data file suitable for input to a circuit simulator. The names used by the designer inside the system are employed in the data file where possible and comments are automatically produced to clarify obscure data file conventions.

5.4.3.4.2 LOGIC LEVEL

A simple logic simulator based on one written by J. Rowson and J. Wipfli [Rowson78] was integrated into the design environment. Interfaces could also be provided to other external simulators. The main advantages of integrating the logic simulator is that the system can provide graphical feedback and allow interactions by the designer driving the logic simulator. Primitive components and block instances can be simulated.

The logic simulator models pulldowns and pass transistors as switches and models pullups as resistors. An electrical node may be assigned one of seven states:

FIXHI - supply of logic 1

ONE - logic 1 sourced by a transistor

WAS1 - logic 1 stored on a capacitor

UNKNOWN

WAS0 - logic 0 stored on a capacitor

ZERO - logic 0 sunk by a transistor

FIXLO - supply of logic 0

The algorithm for simulation consists of four steps:

(i) prediction phase - degrading ONE and ZERO to WAS1 and WAS0

(ii) propagation of FIXHI nodes

(iii) propagation of FIXLO nodes

(iv) resolution of remaining nodes

This is a particularly simple simulator. It contains no information on timing or delays but resolves logic values given a set of inputs and existing states. Its deficiencies are not held to be important in this context. A more extensive simulator could be introduced as an external element of the system c.f. SPICE (Section 5.4.3.4.1). The intention in including this simulator within the system was to illustrate the functionality of the system and provide an example of this type of subsystem organisation.

5.4.3.4.3 BLOCK LEVEL

A block definition is represented by a SIMULA class which can have both procedural and data attributes. One of the procedural attributes can be a definition of its behaviour. Clearly the designer can engage the power of the general purpose language in describing the behaviour of the block, but one constraint is that the interfaces to

its external environment must conform to the pin definitions. One method of describing behaviour is to use a finite state machine. Since each block instance may be in a different state at any given time, the state must be an attribute of the instance and not the definition.

Setting a mode of operation within the system causes the logic simulator to execute the procedural model of the block's behaviour instead of delving into its internal structure of wires and transistors. The procedure consists of a sequence of conditionals and associated actions based on the current state of the particular block instance and the identity and value of the stimulating input. An example of such a procedure is more fully discussed in Section 6.3.3.4 and an encoded simulation procedure is included in Appendix A.

6. IC DESIGN EXAMPLE

This chapter presents an example of the use of the IC design system discussed in Chapter 5. The original intention was to choose a design which was easy to understand but of sufficient complexity to exhibit and test the full capabilities of the system. The example chosen was the Arithmetic Logic Unit (ALU) data path of OM2 designed by D. Johannsen [Mead80]. It is the data processing element of one of the major components of a microprocessor chip set. The SIMULA description of the data path was generated by inspection of the artwork. The code file is contained in Appendix B. Unfortunately, the size of this design swamped the capabilities of DEC-10 SIMULA run-time system and so only the design organisation and plotting subsystem descriptions use OM2 as an example. It was found necessary to fall back on the simpler shift register design of Appendix A to demonstrate the other subsystems.

6.1 ARCHITECTURAL DESCRIPTIONS

Architectural descriptions of both designs are presented in this section. This enables the reader, with the aid of the statistics presented in each subsection, to gauge the complexity of the designs and relate that to the structure and volume of the descriptive code contained in

Appendices A and B. Section 6.2 comments on the organisation of the design description files and section 6.3 contains discussions of the subsystem facilities which use examples from the designs and presume a general familiarity with their architectures. In general, logic and circuit diagrams have been produced manually while stick diagrams and artwork have been produced using ICSYS.

6.1.1 THE SHIFT REGISTER

A single shift register cell contains an inverter coupled to a pass transistor. If a single bit wide register is n bits in length then it may be constructed from 2*n basic blocks. An m-bit wide shift register consists of a vertical stack of m registers. Therefore an m-bit wide register which is n bits in length requires 2*m*n block instances. Data moves through the shift register under the control of the two phase non-overlapping clocks driving the pass transistors. Power and ground run horizontally through the register in metal, data also runs horizontally but in polysilicon between the cells and the clock lines run vertically in polysilicon. Adjacent cells abut to make connections. Lines or small sections of code from Appendix A have already been used as examples in Section 5.3. The logic diagram, stick diagram and artwork of a single shift register cell are shown in Figure 6-1. Similar

representations of a register array are shown in Figure 6-2 which also illustrates the ability of the system to plot at various depths of block instance nesting.

STATISTICS - SHIFT REGISTER CELL (APPENDIX A)

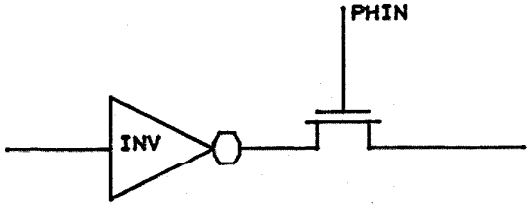
Code starts at line : 126
No. of lines of code : 131
No. of pins : 8
No. of wires : 14
No. of transistors : 3
No. of block instances : 0
No. of coordinodes placed : 22

The code module describing the shift register cell also contains a procedural description of its behaviour which occupies 35 lines of code, starting at line 131.

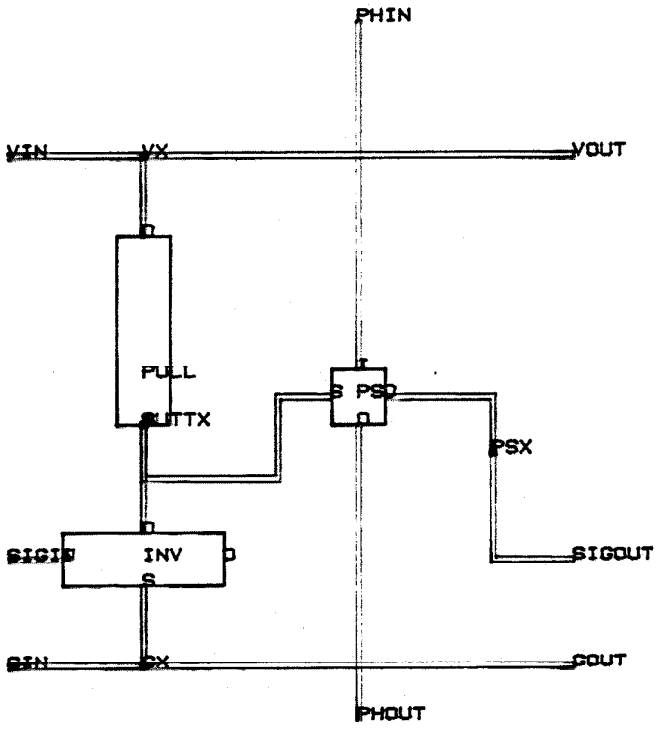
STATISTICS - SHIFT REGISTER ARRAY (APPENDIX A)

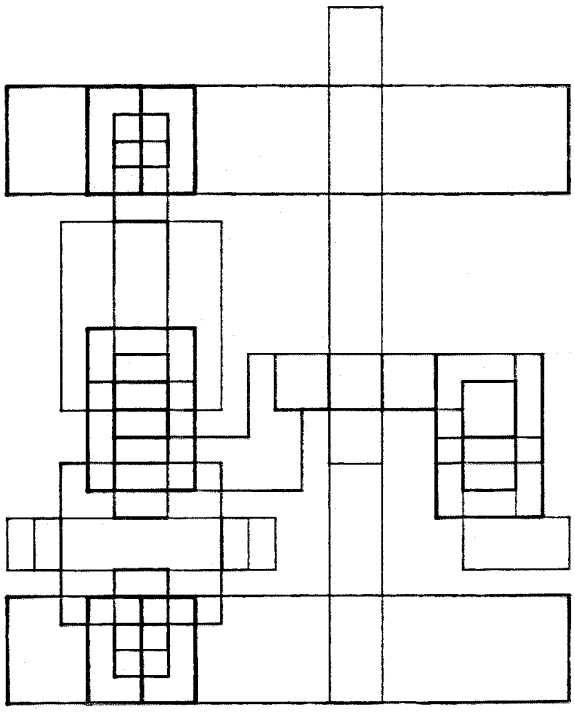
Code starts at line : 262
No. of lines of code : 96
No. of pins : 10
No. of wires : 14
No. of transistors : 0
No. of block instances : 1
No. of coordinodes placed : 10

These counts relate to the number of statements of the specified type and not to the number of times each statement is executed since some statements occur inside loops and are executed a variable number of times dependent on the parameterisation of the block.



(A) LOGIC DIAGRAM

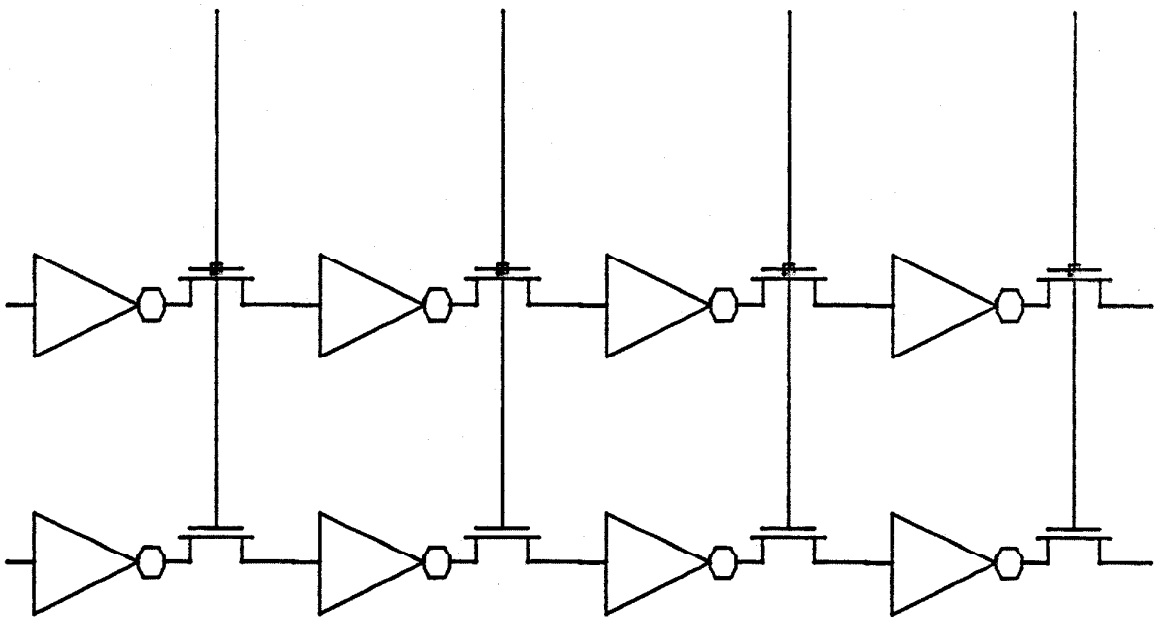




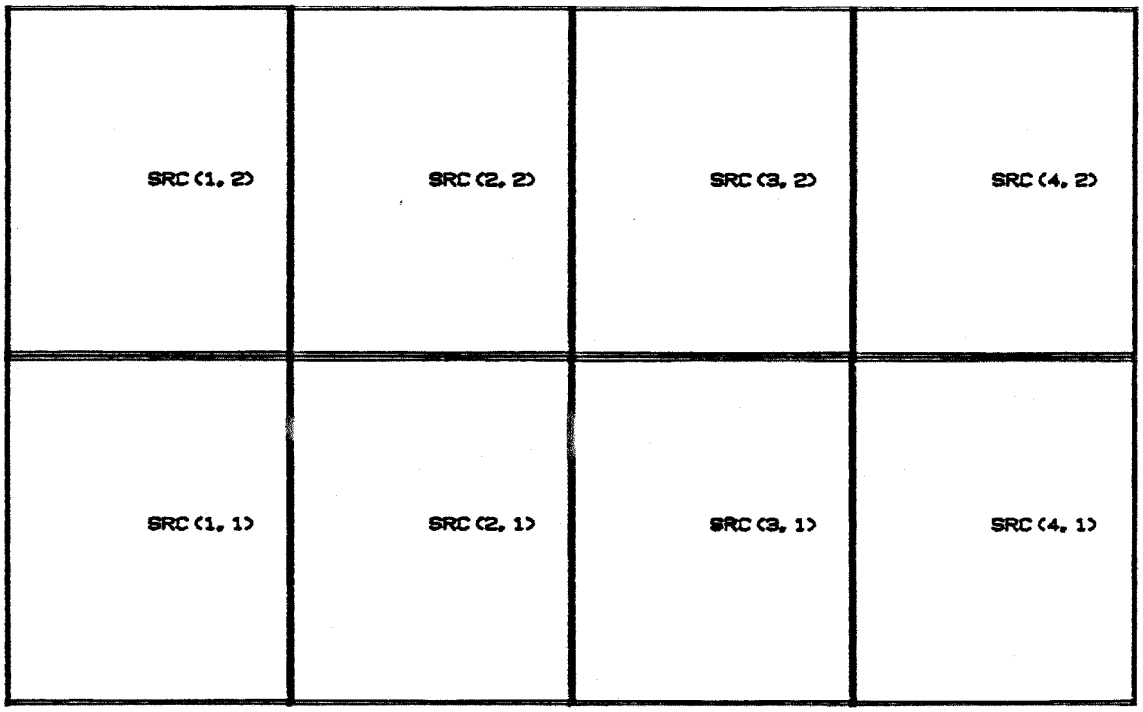
(B) STICK DIAGRAM

(C) LAYOUT

FIGURE 6-1. SHIFT REGISTER CELL.

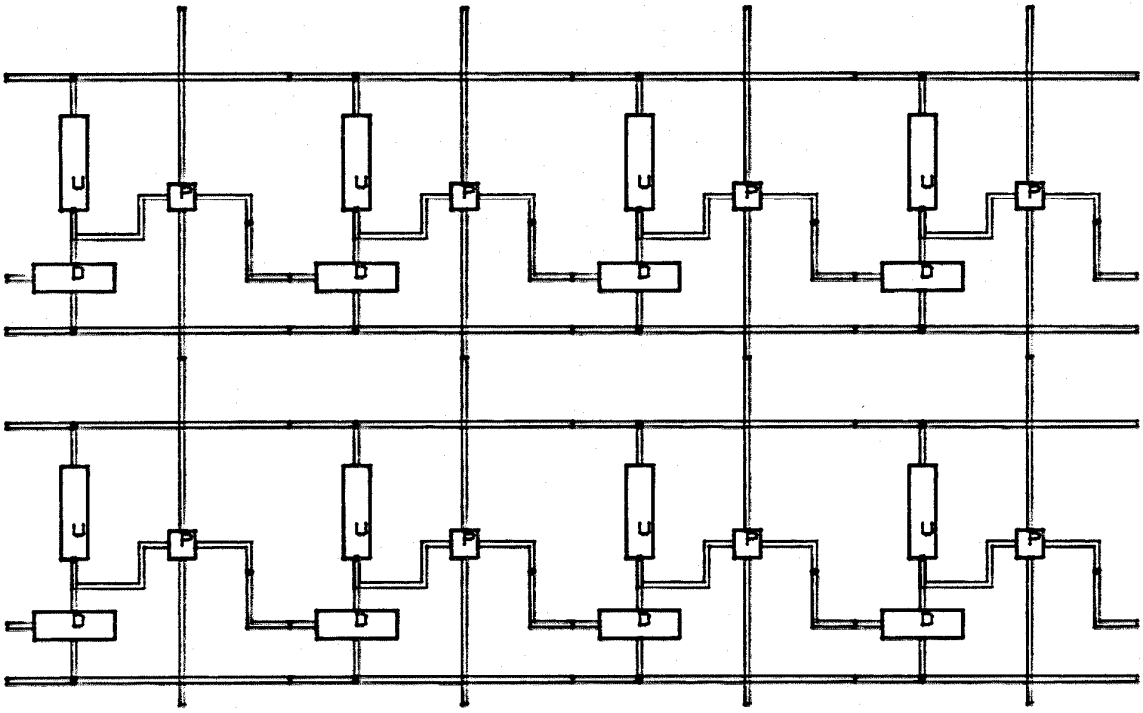


(A) LOGIC DIAGRAM

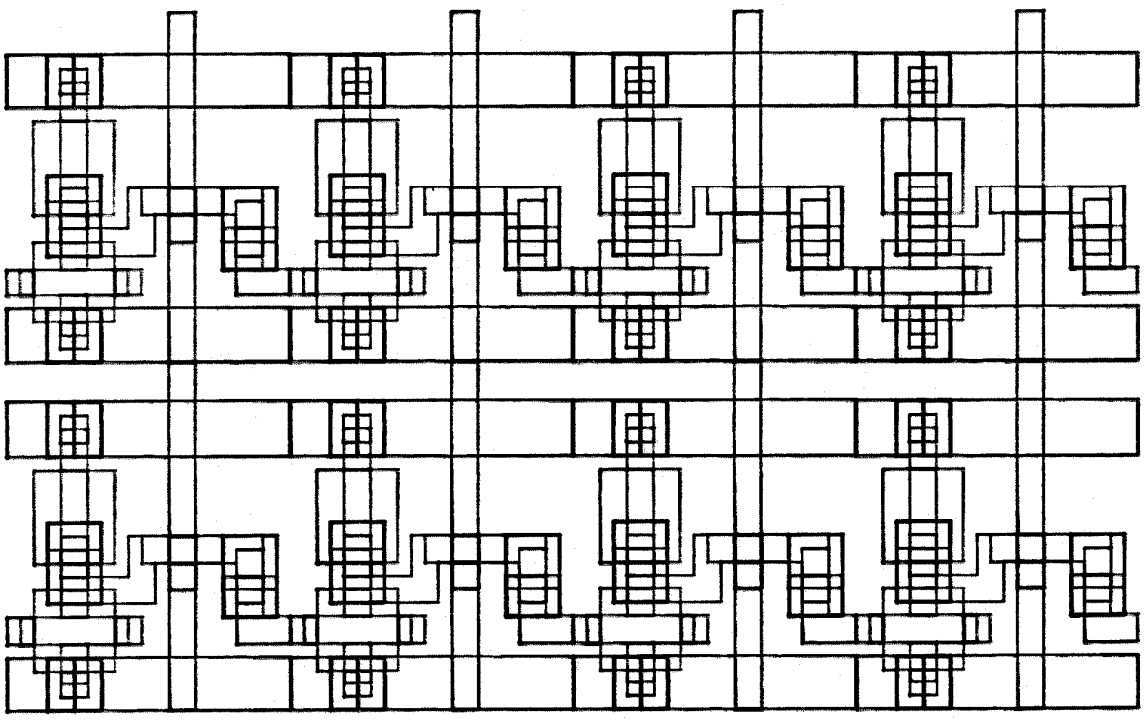


(B) BLOCK DIAGRAM

FIGURE 6-2. SHIFT REGISTER ARRAY.



(C) STICK DIAGRAM



(D) LAYOUT

FIGURE 8-2. SHIFT REGISTER ARRAY.

6.1.2 THE DATA PATH

The architecture of the data path conforms to a traditional pattern of data flow. The layout of an earlier version of the complete OM2 data path chip appears as the frontispiece of [Mead80] and is reproduced in Figure 6-3. The description of the data path included in this chapter deals only with the central functional block of the chip which performs the data manipulation operations of the processor under the direction of control microinstructions.

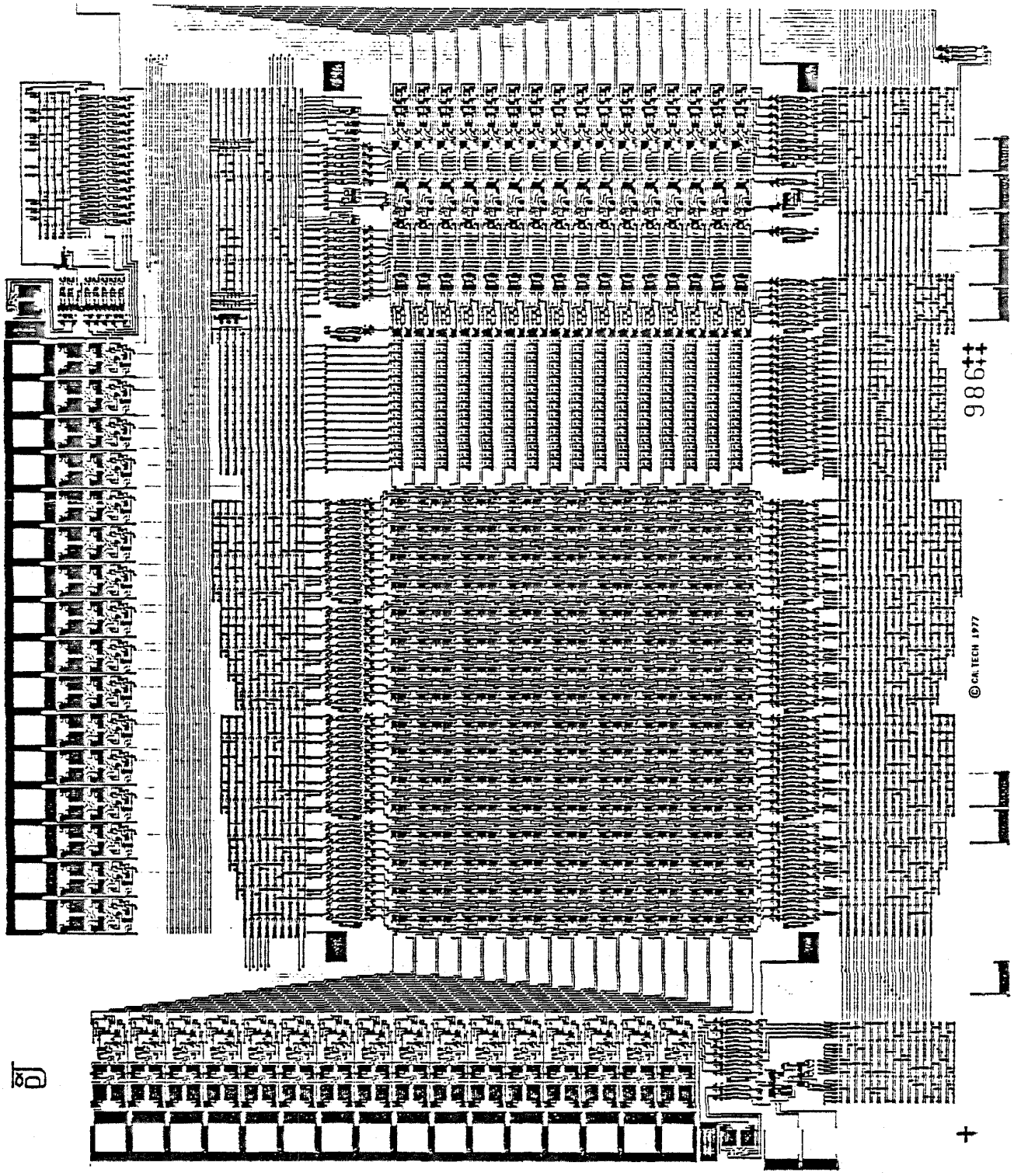


FIGURE 6-3. OM2 DATA PATH CHIP.*

* REPRODUCED BY PERMISSION OF PROF. C. A. MEAD

The data path has two internal busses which run horizontally for the length of the ALU in metal. Power and ground run parallel to the bus lines, also in metal. Control lines run perpendicularly to the data flow in polysilicon and follow a two phase (phi1,phi2) non-overlapping clock scheme.

The data path is constructed from four basic block types, each with the same bus structure. They have been designed so that they may be stretched to identical pitch and slotted together to build a particular formation within the general architectural philosophy. To explain further, at its most dense each block is a different height though the vertical ordering of the busses, which run horizontally through each block, is the same. The blocks are parameterised to stretch in height. The four cell types are the memory, input, carry and output stages. Manually laid out logic diagrams (some with pass transistors) plus annotated stick diagrams and geometric layout produced by the design system are shown in figures accompanying each section.

6.1.2.1 MEMORY

The dual port memory block takes input from, and sends output to, either of the two busses during phi1. The cell is refreshed during phi2. Both the input and its complement are available as required by the input stage of the ALU. Two versions of the cell are possible to cater for two operands connecting to the next stage. The version which abuts the input stage has to carry additional cross-overs. The logic diagram, stick diagram and artwork of the basic memory block are shown in Figure 6-4.

STATISTICS - MEMORY STAGE CELL (APPENDIX B)

Code starts at line : 145
No. of lines of code : 264
No. of pins : 28
No. of wires : 52
No. of transistors : 9
No. of block instances : 0
No. of coordinodes placed : 73

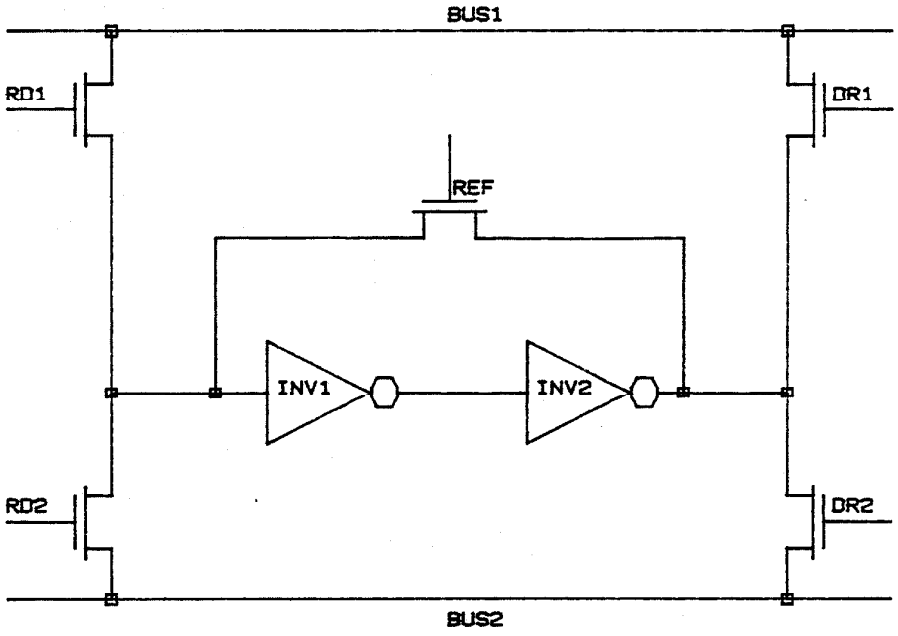


FIGURE 6-4(A). MEMORY CELL - LOGIC DIAGRAM.

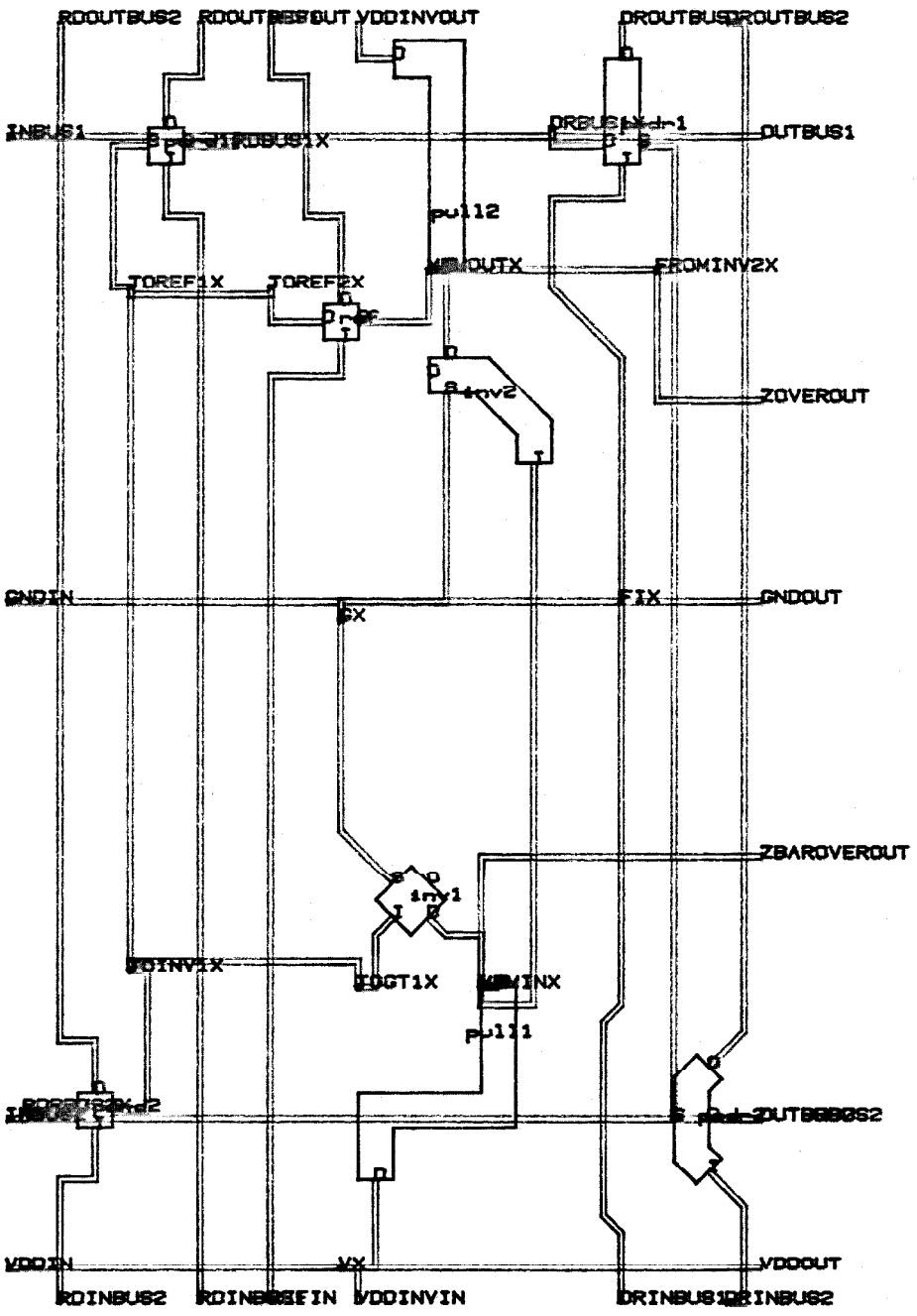


FIGURE 6-4(B). MEMORY CELL - STICK DIAGRAM.

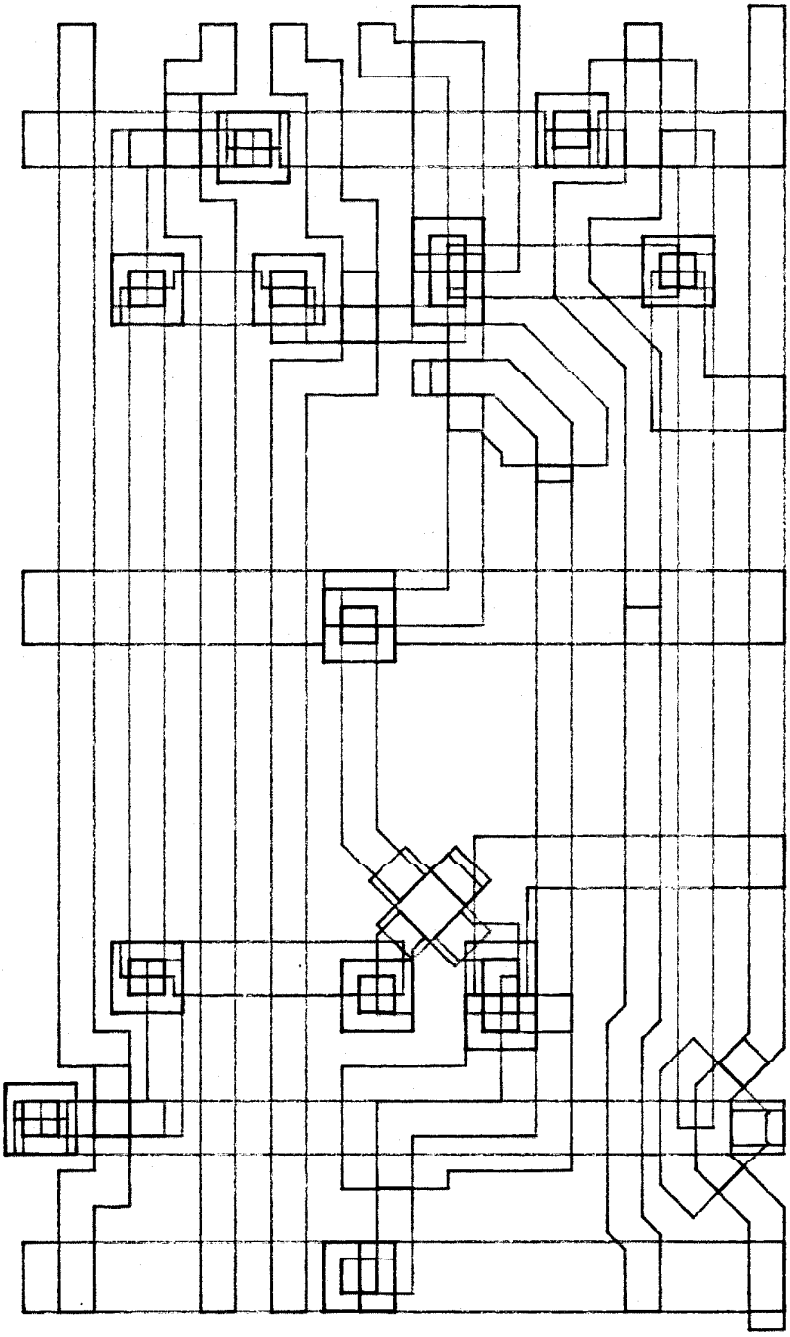


FIGURE 6-4(C). MEMORY CELL - LAYOUT.

6.1.2.2 INPUT

The purpose of the input stage is to generate the carry-propagate and carry-kill signals from the memory outputs and their complements under the control of microinstruction inputs. First all combinations of the inputs are generated then logical functions are selected for each of the propagate or kill outputs using the two banks of interspersed control lines representing the terms of the propagate and kill functions. The control lines are named according to their function. Each line has an input and output terminal at coordinodes "xxxIN" and "xxxOUT" respectively. They are positioned so that multiple instances of the same block may be stacked vertically. Carry-kill control line inputs and outputs begin with the letter K and are of the form "KxxIN" and "KxxOUT". Similarly carry-propagate control line inputs and outputs begin with the letter P and are of the form "PxxIN" and "PxxOUT". The remaining two letters of each name are permutations of F and T, representing the inverted and uninverted inputs A and B which appear at the left of the block. Therefore if the "KFTIN" - "KFTOUT" line is enabled then the value of AND ("ABARIN","BIN") is selected as the output on "KBAR". The logic diagram, stick diagram and artwork of the input stage block are shown in Figure 6-5.

STATISTICS - INPUT STAGE CELL (APPENDIX B)

Code starts at line : 415

No. of lines of code : 339

No. of pins : 32

No. of wires : 75

No. of transistors : 18

No. of block instances : 0

No. of coordinodes placed : 122

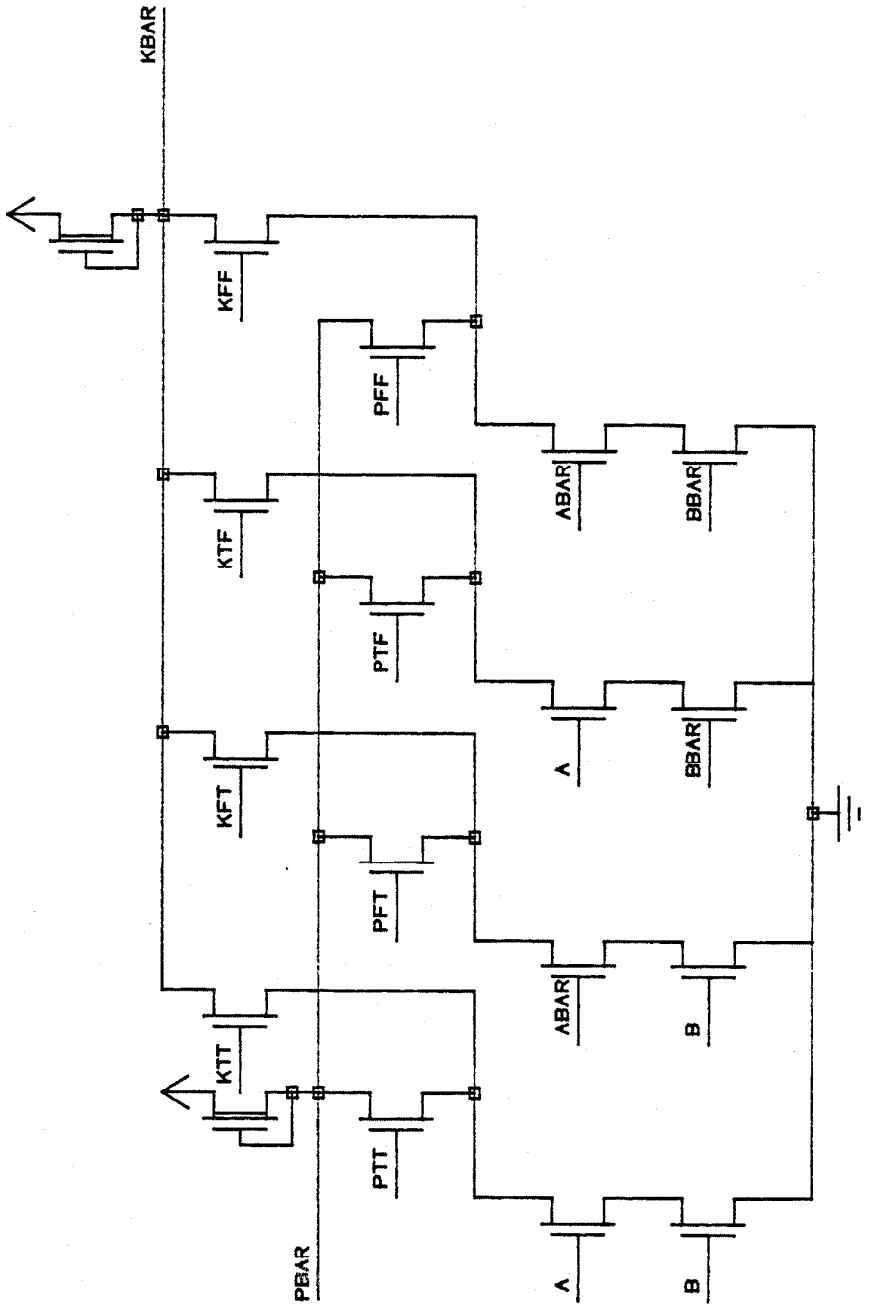


FIGURE 6-5(A). INPUT CELL - LOGIC DIAGRAM.

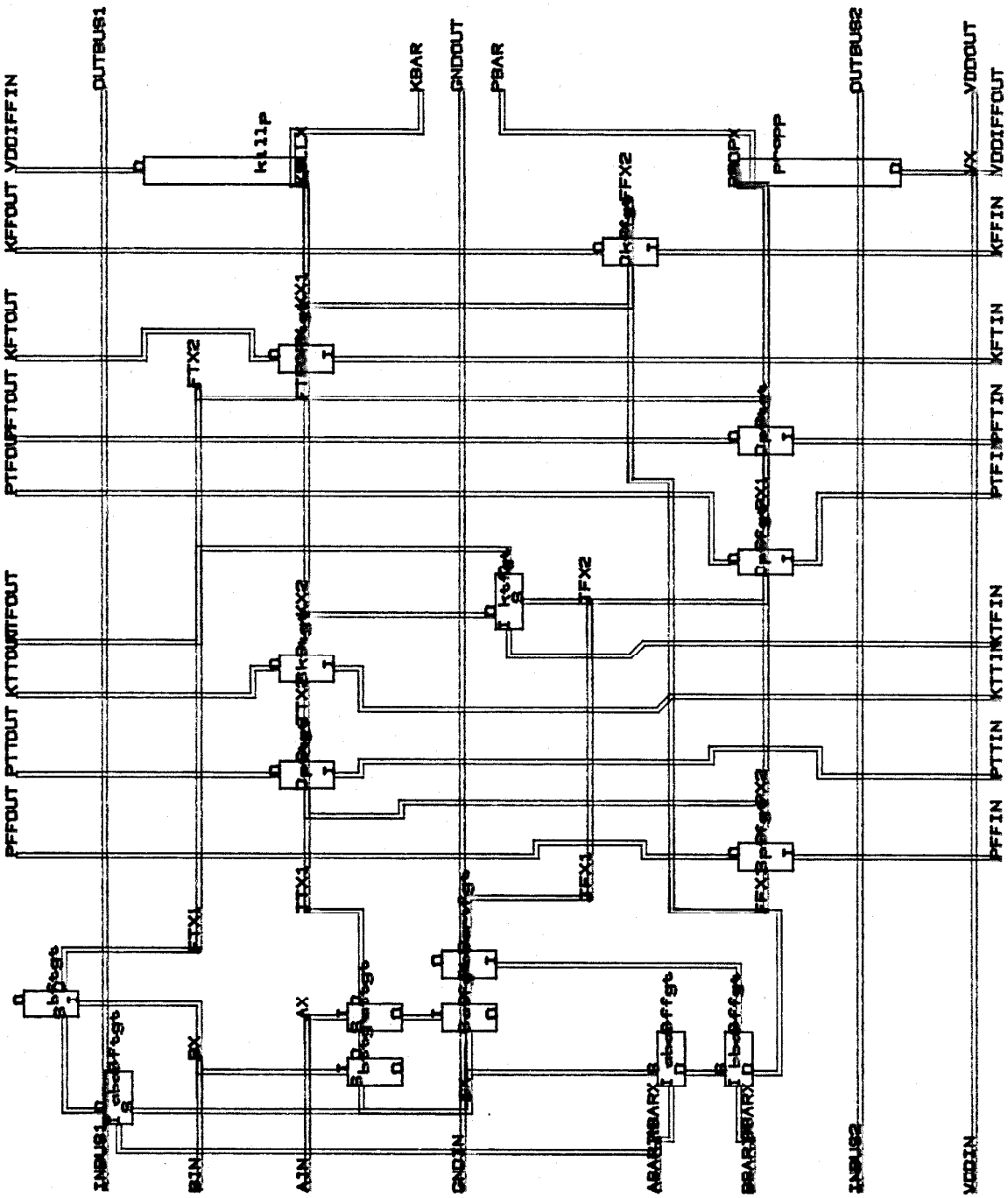


FIGURE 6-5(B). INPUT CELL - STICK DIAGRAM.

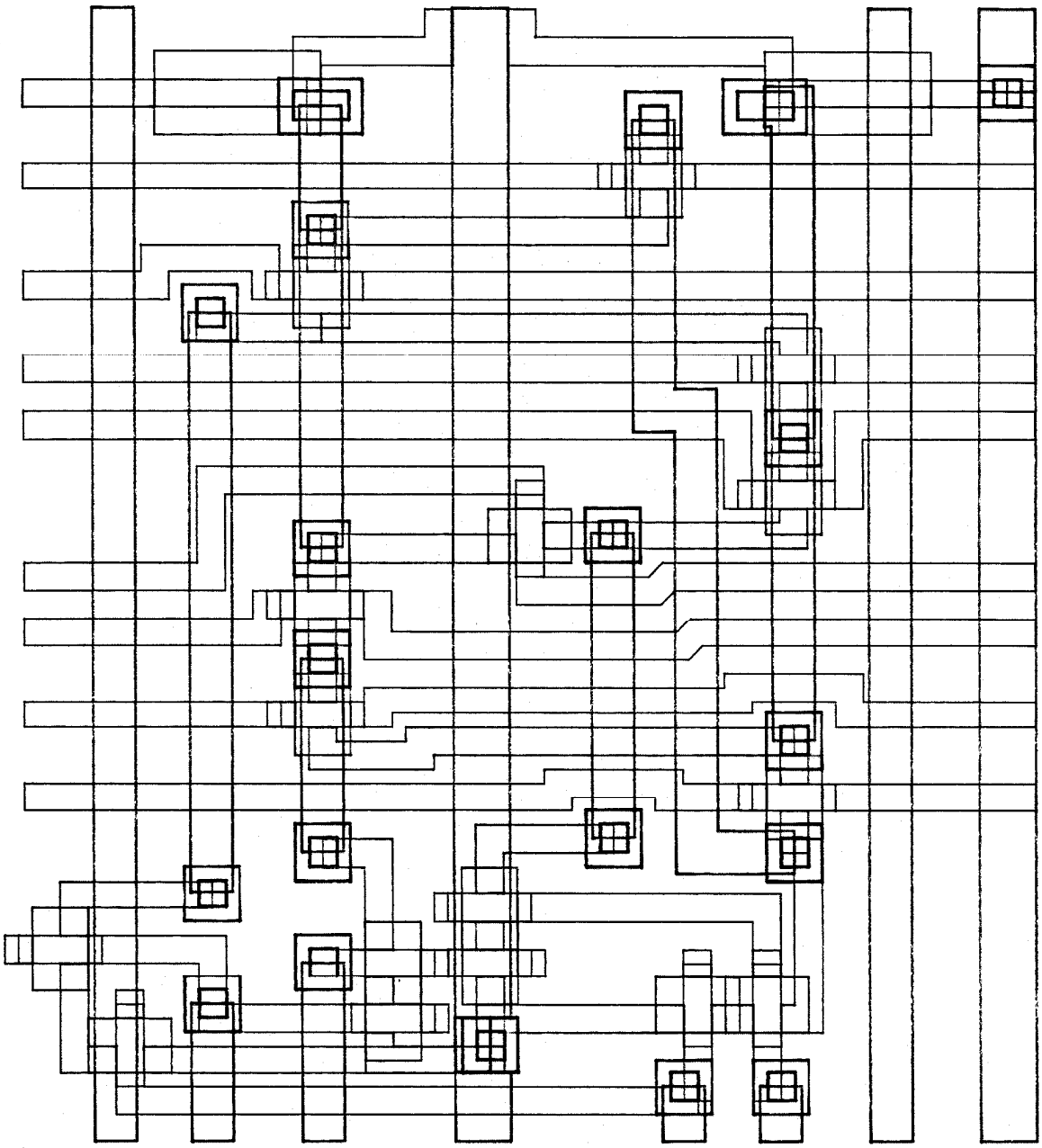


FIGURE 6-5(C). INPUT CELL - LAYOUT.

6.1.2.3 CARRY

The carry stage of the ALU takes the propagate and kill signals from the input stage and the immediately lower abutting cell's carry-out to produce a carry-out for the next higher block and the propagate and carry-out and their inversions for the output stage. Carry-out is precharged to speed up the operation of the block. There are two versions of the carry stage. In one the carry-in signal is directed through an amplification stage to restore the signal level; in the other the direct vertical connection is made. The logic diagram, stick diagram and artwork of the carry stage block are shown in Figure 6-6.

STATISTICS - CARRY STAGE CELL (APPENDIX B)

Code starts at line : 760

No. of lines of code : 273

No. of pins : 25

No. of wires : 60

No. of transistors : 14

No. of block instances : 0

No. of coordinodes placed : 91

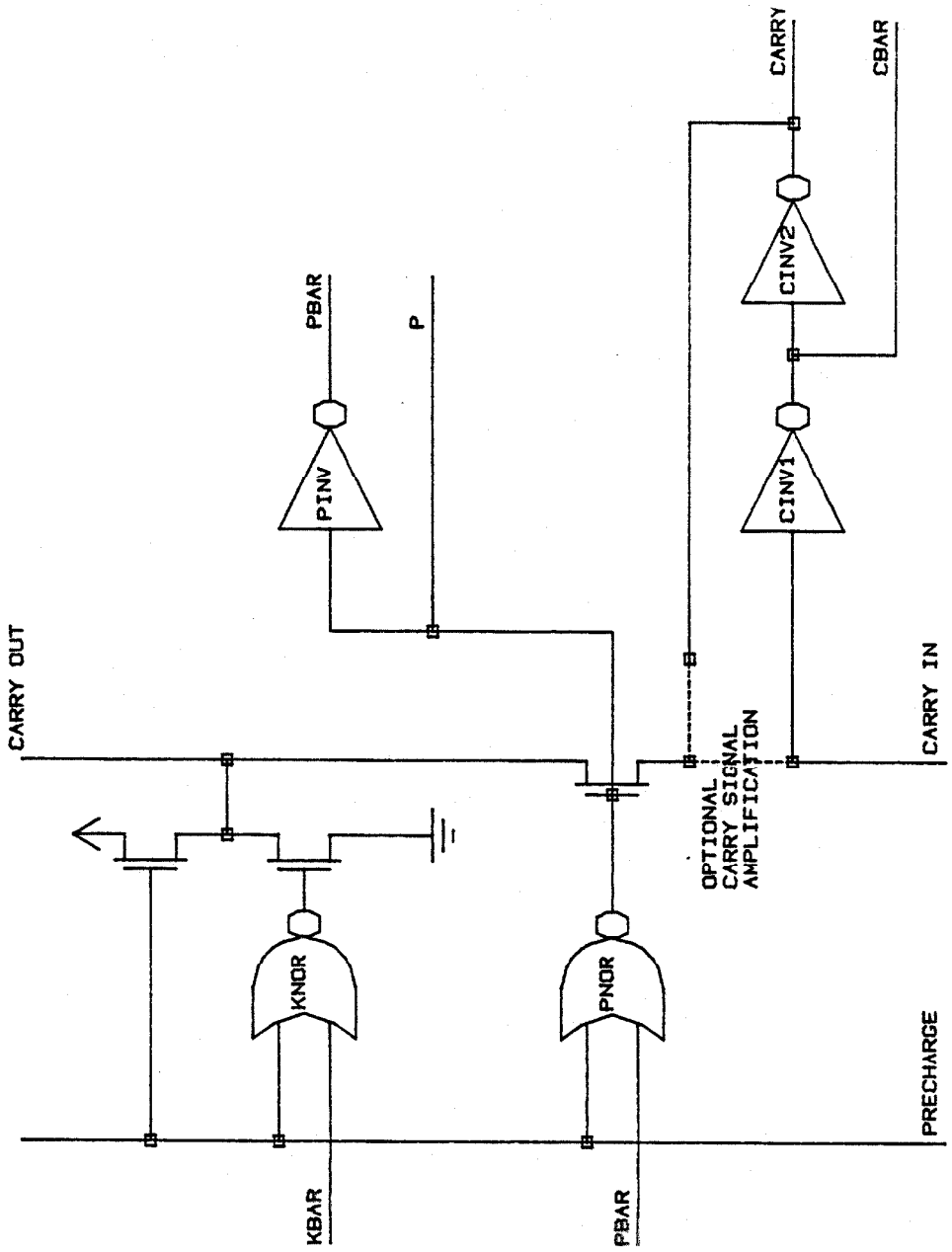


FIGURE 6-6(A). CARRY CELL - LOGIC DIAGRAM.

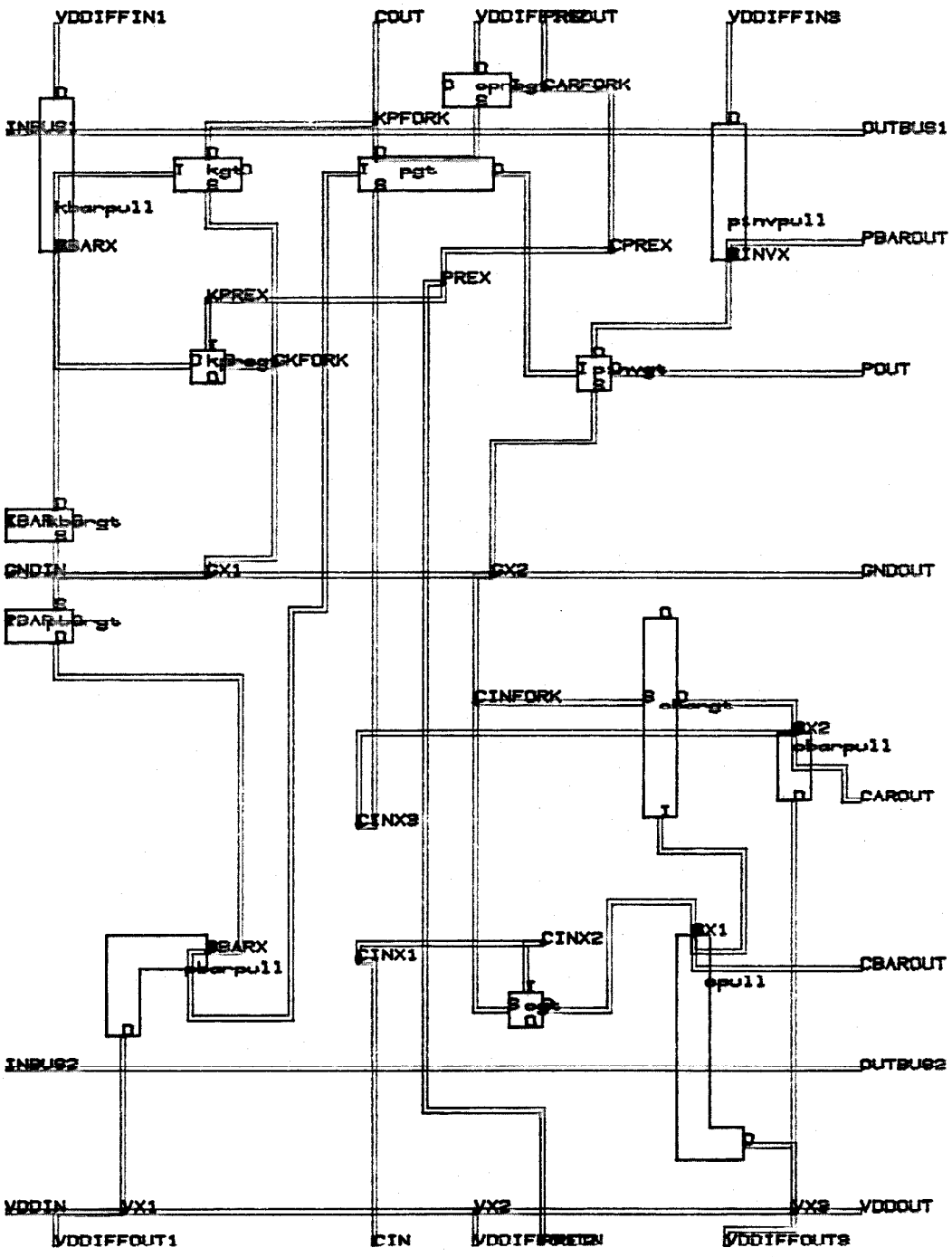


FIGURE 6-6(B). CARRY CELL - STICK DIAGRAM.

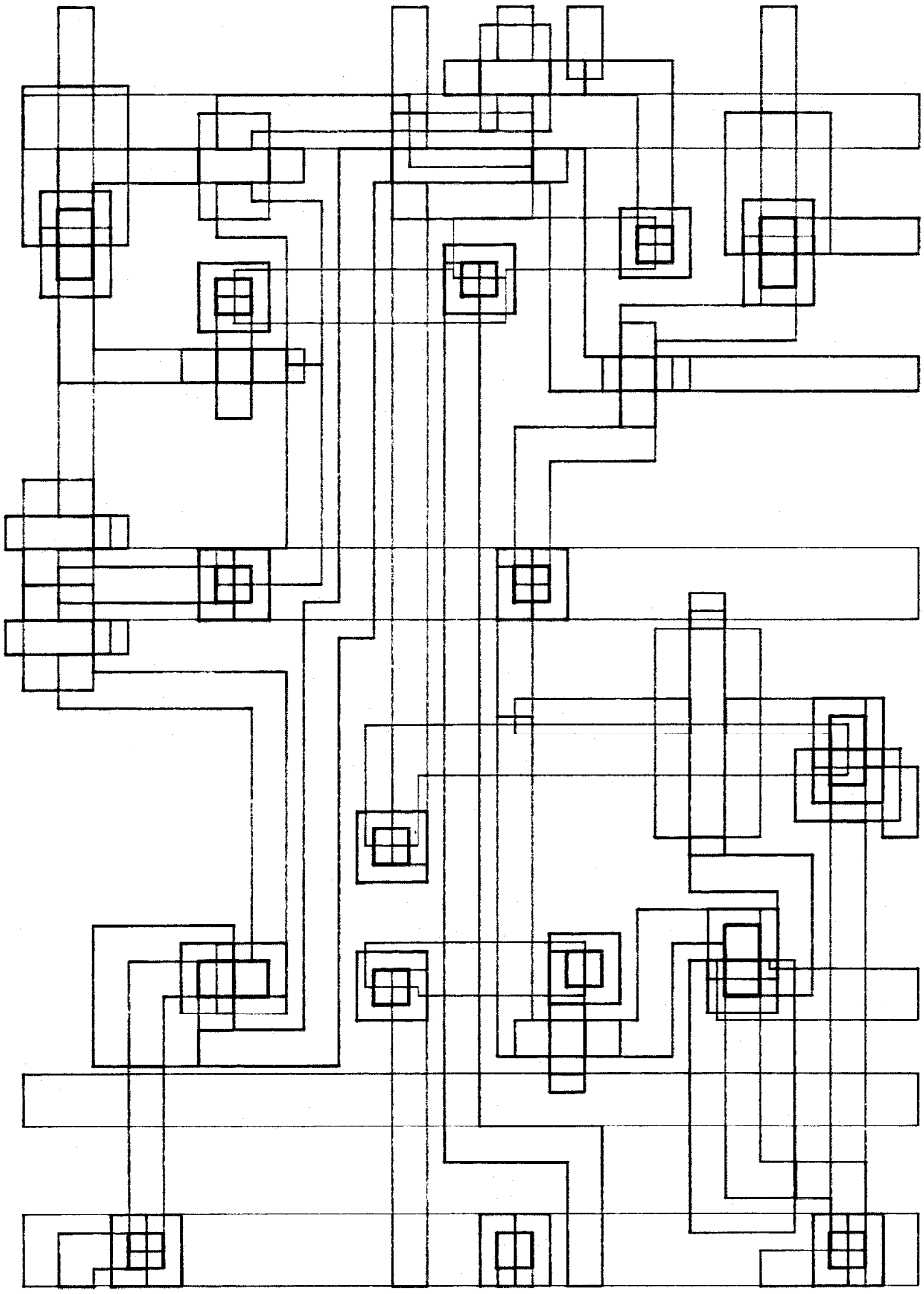


FIGURE 6-6(C). CARRY CELL - LAYOUT.

6.1.2.4 OUTPUT

The output stage of the ALU takes the propagate and carry signals, and their inversions, and first forms all possible logical combinations of these signals. These are selected by functions of the four control lines and output onto either of the busses. The result control lines are encoded in the same way as the carry-propagate and carry-kill control lines of the input stage. The logic diagram, stick diagram and artwork of the output stage are shown in Figure 6-7.

STATISTICS - OUTPUT STAGE CELL (APPENDIX B)

Code starts at line : 1039

No. of lines of code : 329

No. of pins : 26

No. of wires : 75

No. of transistors : 17

No. of block instances : 0

No. of coordinodes placed : 119

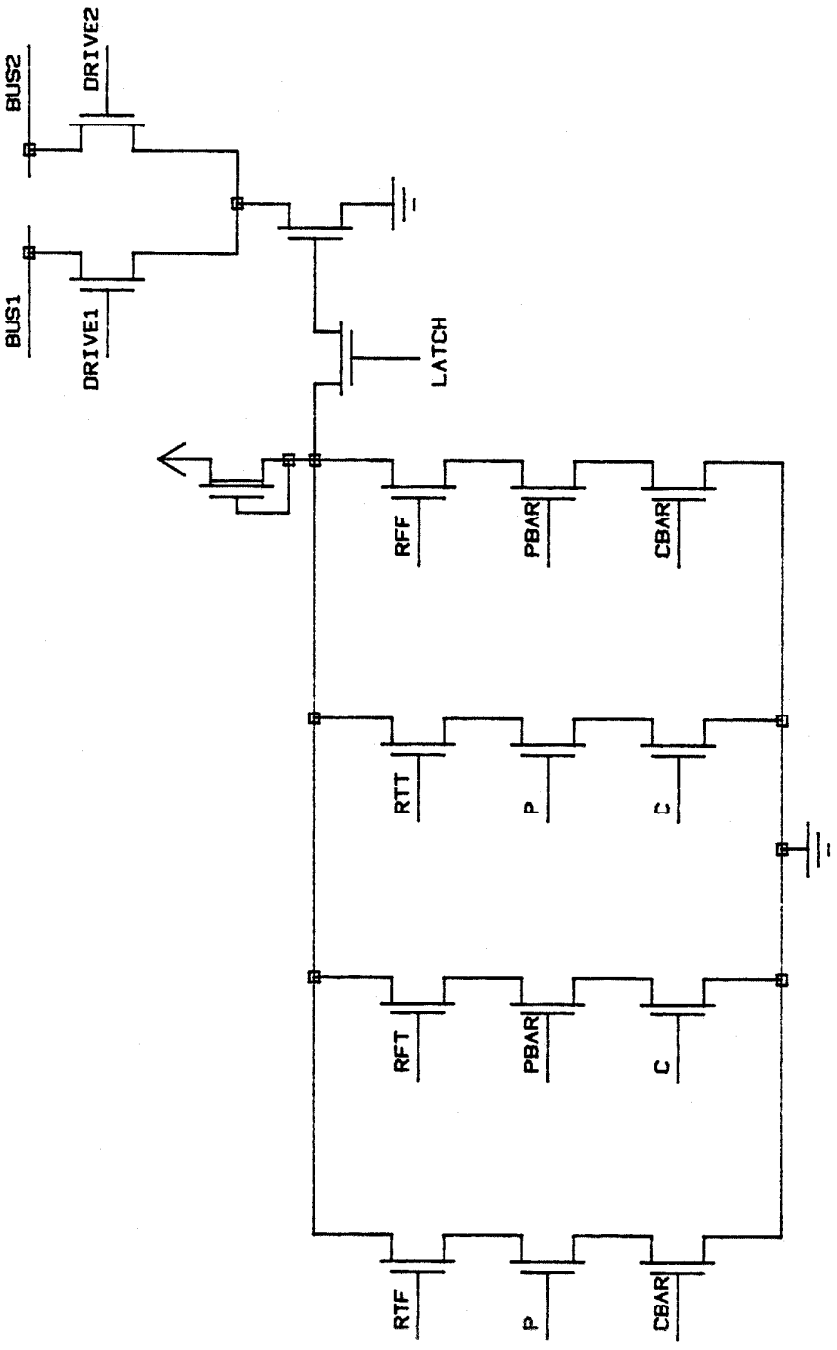


FIGURE 8-7 (A). OUTPUT CELL - LOGIC DIAGRAM.

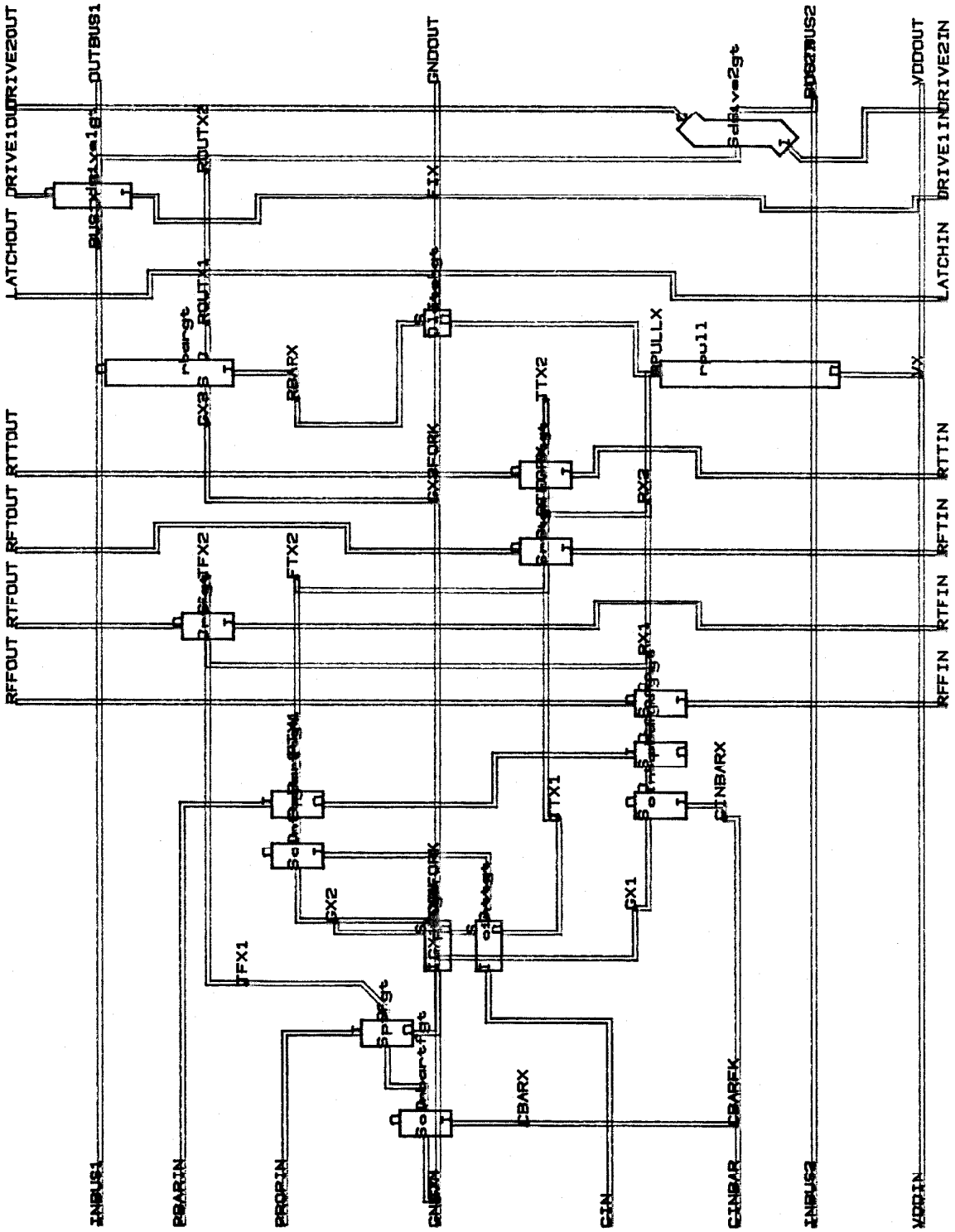


FIGURE 6-7(B). OUTPUT CELL - STICK DIAGRAM.

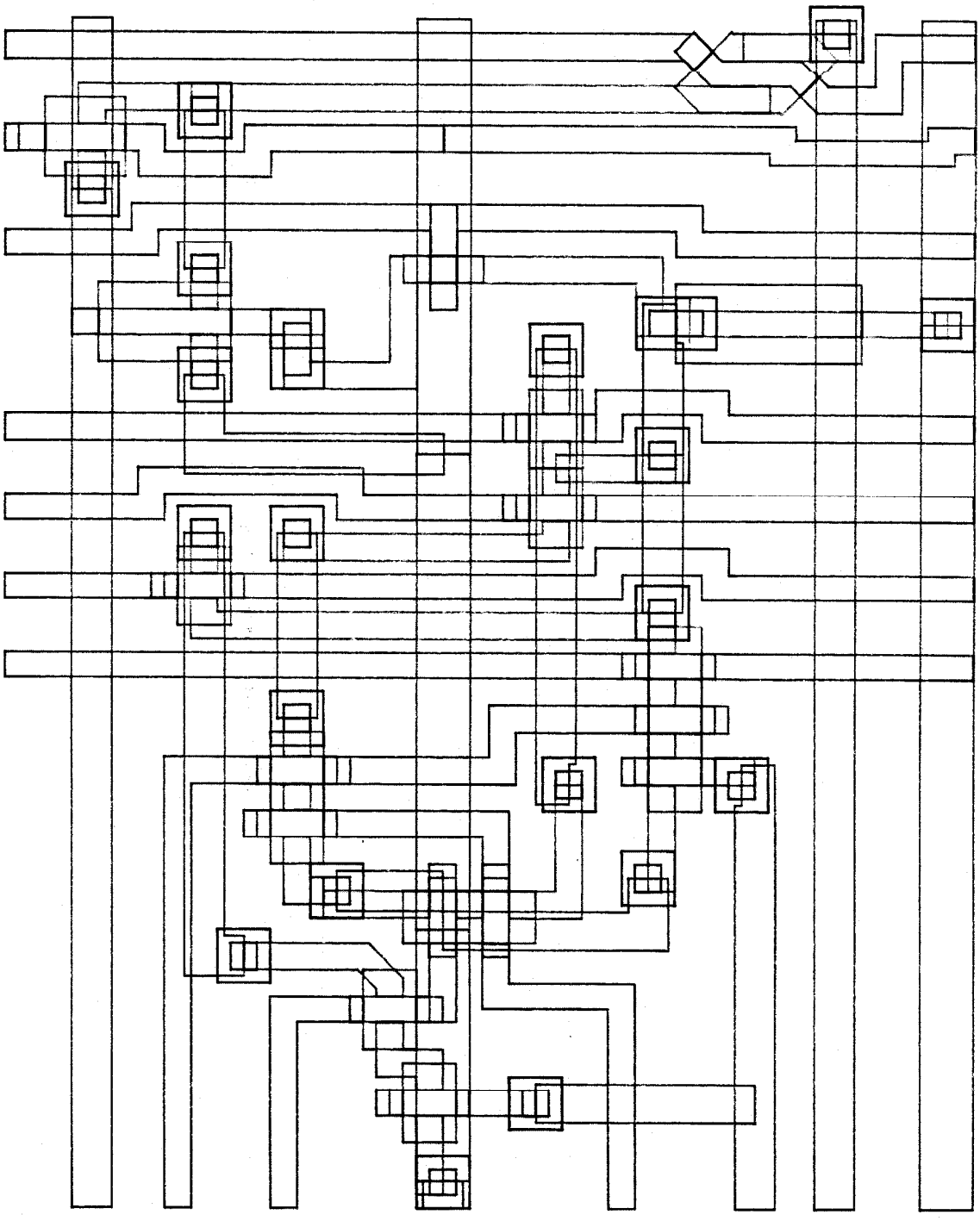


FIGURE 6-7(C). OUTPUT CELL - LAYOUT.

6.1.2.5 ALU BIT SLICE

A single bit data path through one configuration of the ALU can be built from a sequence of two memory, one input, one carry and one output block. The different versions of the carry stage can be used to make two different versions of the bit slice. The amplified version is used for one bit slice in four. The floor plan of a bit slice is shown in Figure 6-8.

STATISTICS - ALU BIT SLICE (APPENDIX B)

Code starts at line : 1374
No. of lines of code : 343
No. of pins : 70
No. of wires : 99
No. of transistors : 0
No. of block instances : 4
No. of coordinodes placed : 78

The organisation of the code in this module has been complicated by the attempts to build part slices which would fit into the space available under DEC-10 SIMULA.

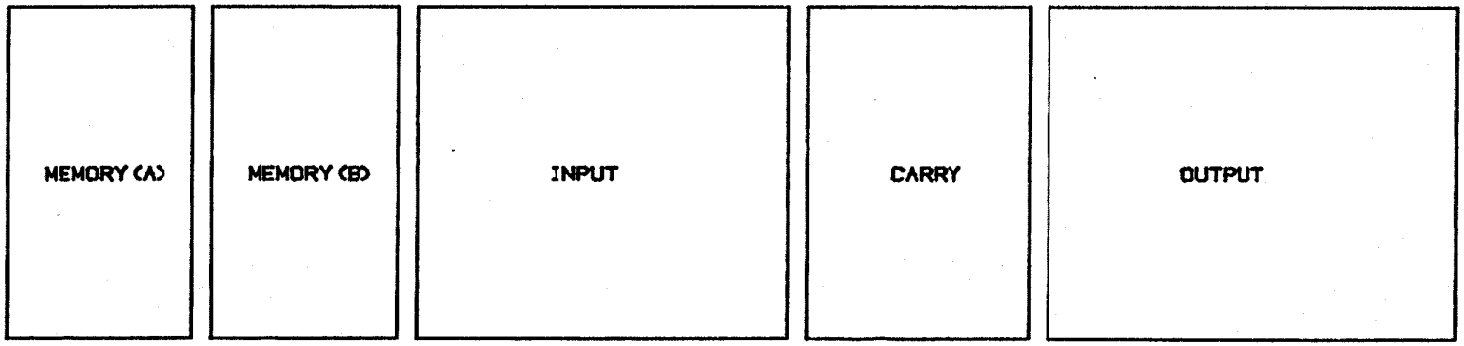


FIGURE 6-8. ALU BIT SLICE.

To illustrate the complete function of the data path, consider the operation of addition [Mead 80]. Recall the names and encoding of the control lines as explained previously. If both inputs from the memory cells contain a 1, the carry is to be generated while if both inputs are 0 the carry is killed. If the two inputs differ the carry is to be propagated (carry out <- carry in). To do this operation the kill output should be active if both inputs are false, so KFF is enabled. Both PFT and PTF should be enabled to propagate properly. Therefore, K=(KFF,KFT,KTF,KTT)=(1,0,0,0) and P=(PFF,PFT,PTF,PTT)=(0,1,1,0). The result of the ALU operation is produced by the output function block. For addition the output should be the exclusive-or of propagate and carry, so R=(RFF,RFT,RTF,RTT)=(0,1,1,0). The output can be directed onto either of the busses.

6.1.2.6 ALU DATA PATH

To construct a 4-bit wide data path all that is required is to stack one amplified bit slice followed by three ordinary slices. An arbitrarily wide path can be built from the basic slice or a multiple of 4 wide path can be built by stacking 4-bit wide blocks. The floor plan of a 4-bit wide path is shown in Figure 6-9. Note that an extra power rail is needed at the top of the path c.f. Figure 6-13.

STATISTICS - ALU DATA PATH (APPENDIX B)

Code starts at line : 1723
No. of lines of code : 360
No. of pins : 64
No. of wires : 80
No. of transistors : 0
No. of block instances : 1
No. of coordinodes placed : 70

The organisation of the code in this module has been complicated by the attempts to build partial paths which would fit into the space available under DEC-10 SIMULA.

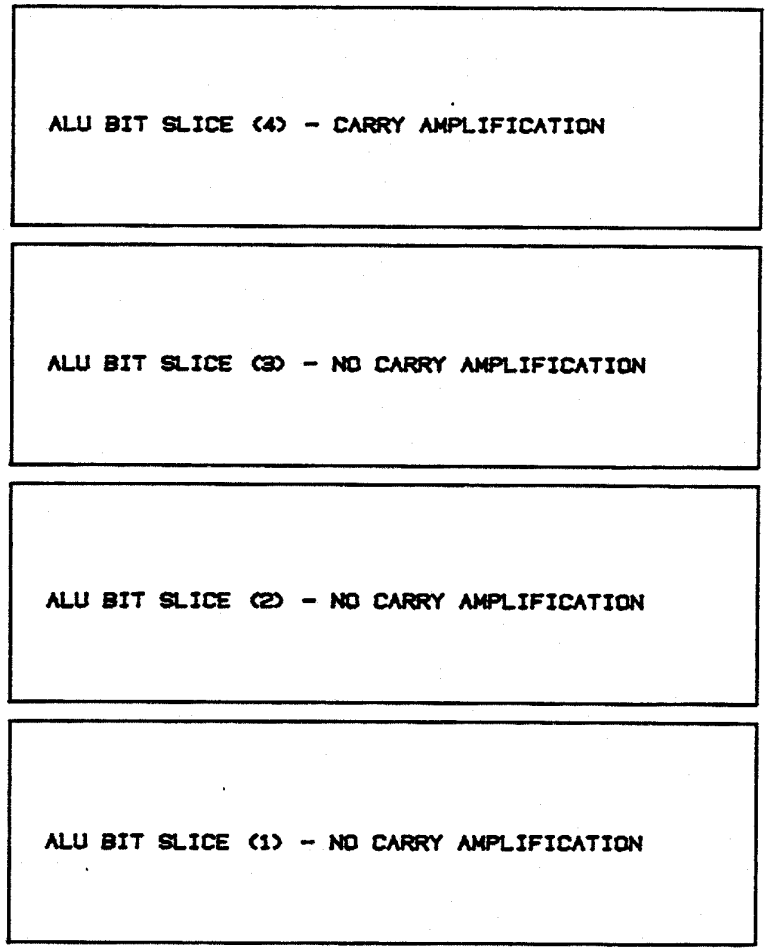


FIGURE 6-9. ALU DATA PATH.

6.2 THE DESIGN DESCRIPTION FILE

The design description file is a prefixed block which is the last in a series of SIMULA class definitions i.e. it lies on top of the IC design environment. The file contains all the information particular to that design including the specialised user interface and class definitions for each of the modules in the design. The designer parameterises each block to the extent that satisfies the needs of the architecture. For example, parameters may include boolean variables to control the selection of alternative structures or integers to define the extent of arrays of cells. The reader may observe instances of these effects by examining Appendix B in relation to the architectural descriptions of the previous section.

A systematic organisation of the description of each block is essential to enable the designer to control the volume of data in the design description. A graphical aid to speed up this process is proposed in Section 7.1. The design examples in the appendices exhibit a top-down, structural before physical, sequence in the design description. This results in sets of procedure calls, each dealing with one aspect of the design. First come the pin definitions (external interface), followed by the components i.e. transistors, wires and contacts

(structural) and finally the positions of the coordinodes (physical). The structural existence of a coordinode is deduced by the system from its being referenced by a pin component or wire. Block instances cause the implicit addition of coordinode connect points to the calling environment. Full use of the loop construct can be made in defining, connecting and positioning array structures.

6.3 SUBSYSTEMS

In this section each of the subsystems previously discussed in Section 5.4 with respect to its implementation is demonstrated using one of the design examples.

6.3.1 GRAPHICAL FEEDBACK

Graphical feedback may be by graphics display terminal or through plotting. The first of these alternatives cannot be properly demonstrated within the confines of this text and therefore only plotting examples are included. Artwork and stick diagram layouts are available from primitive block designs. Memory cell plots of these different forms for a minimum pitch cell are shown in Figure 6-10. Note the differences between Figures 6-4 and 6-10. Figure 6-4 represents a much taller cell than Figure 6-10.

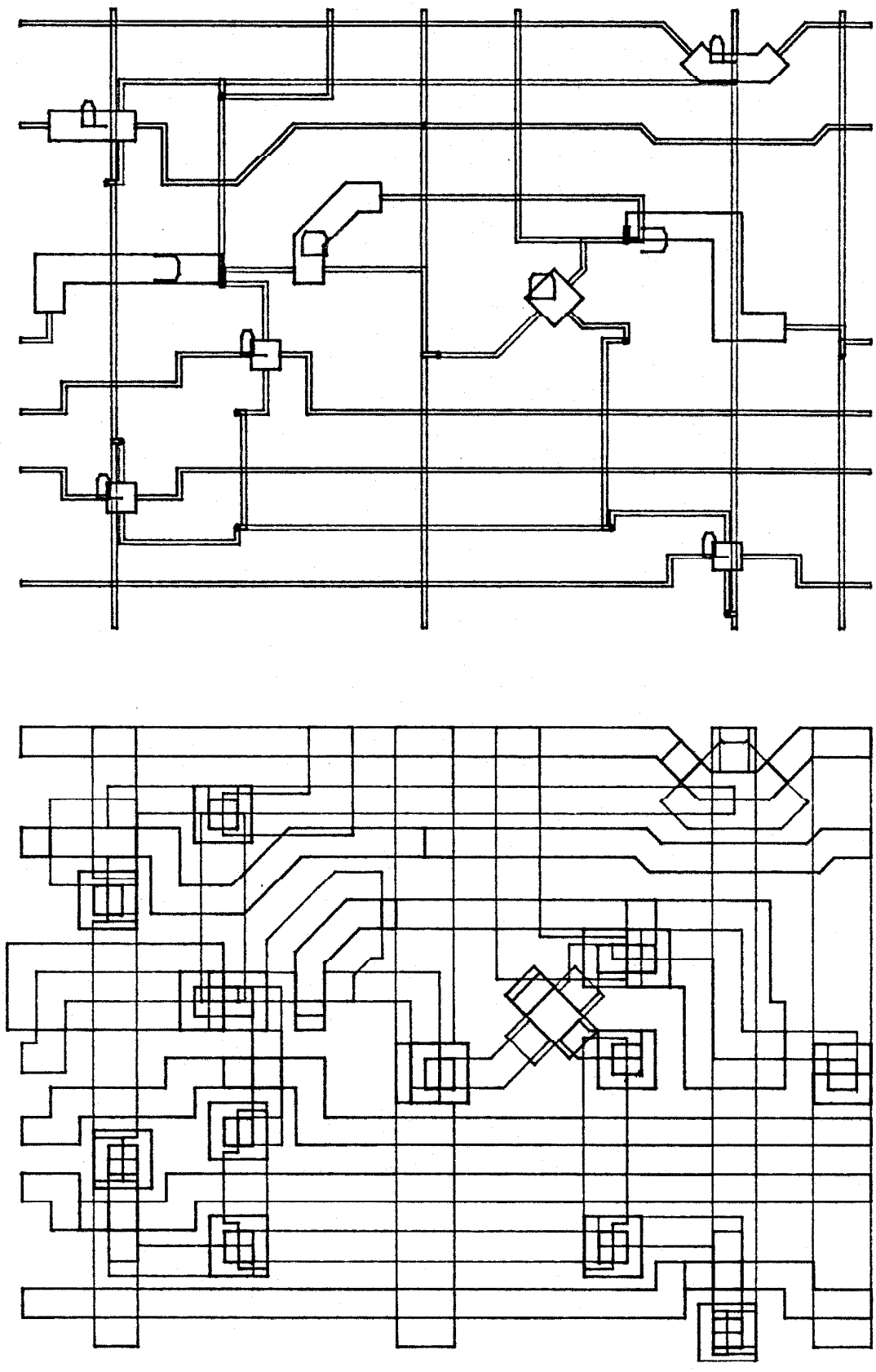


FIGURE 6-10. MEMORY CELL - MINIMUM PITCH.

By parameterisation a memory cell can be constructed which fits the pitch of the rest of the data path. A data path slice normally contains two memory cells which between them supply the four data inputs to the next stage. The two alternative memory cell designs can also be achieved by parameterisation. Their respective stick diagrams are shown connected and abutting in Figure 6-11, which describes a short slice built from only these two cells.

A single bit data path slice contains five block instances i.e. input, carry and output stages following the two memory stages. Unfortunately the plot of this layout could not be constructed due to the familiar deficiencies in the SIMULA run-time system. Therefore two sub-slices of the memory cell pair and a shorter data path, consisting of only the last three stages were built. Their stick diagrams are shown in Figures 6-11 & 6-12.

The highest hierarchical level in the design of the OM2 data path is the n-bit wide data path. Unfortunately this was out of range of the system. It was found that the largest structure which could be built was a 4-bit wide path of slices containing only the two memory stages. Its stick diagram is shown in Figure 6-13.

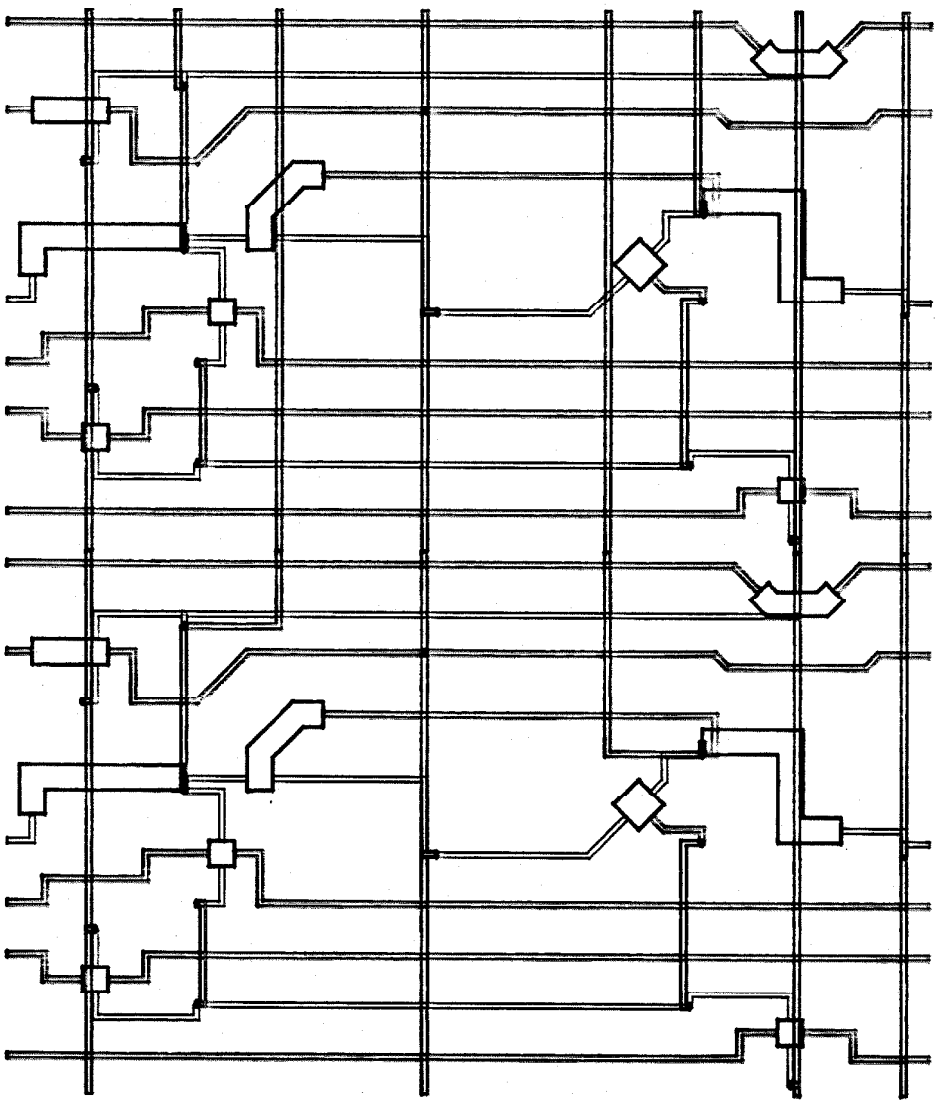


FIGURE 6-11. MEMORY CELLS - TWO STAGE SLICE.

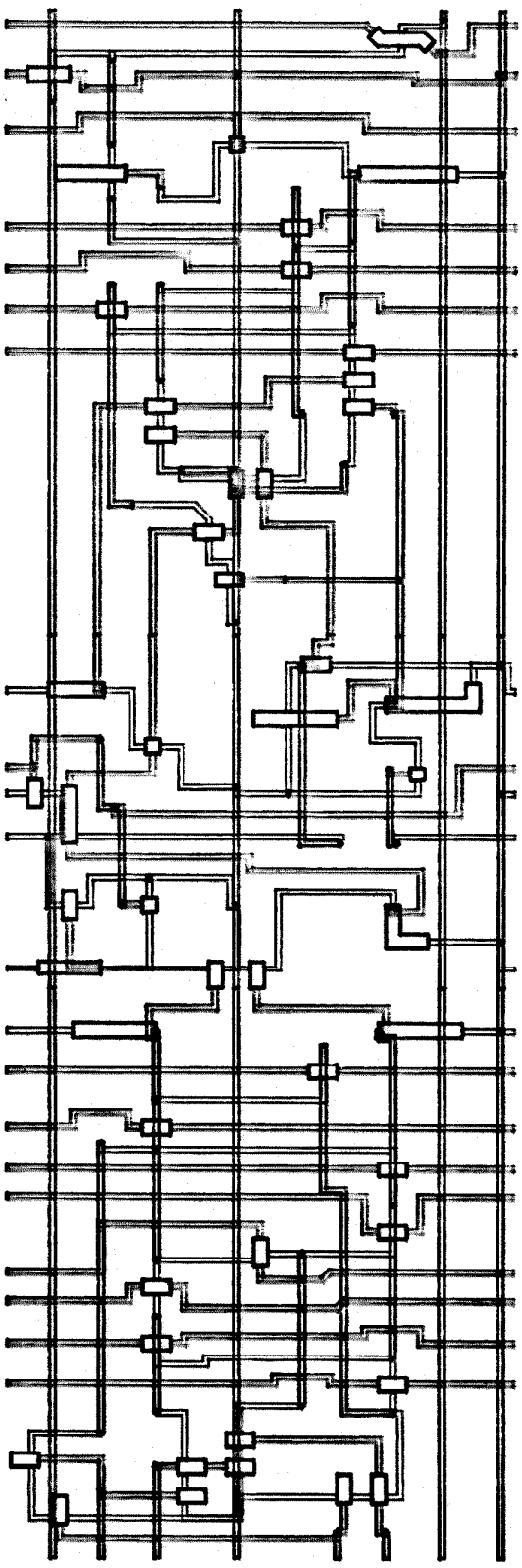


FIGURE 6-12. INPUT + CARRY + OUTPUT SLICE.

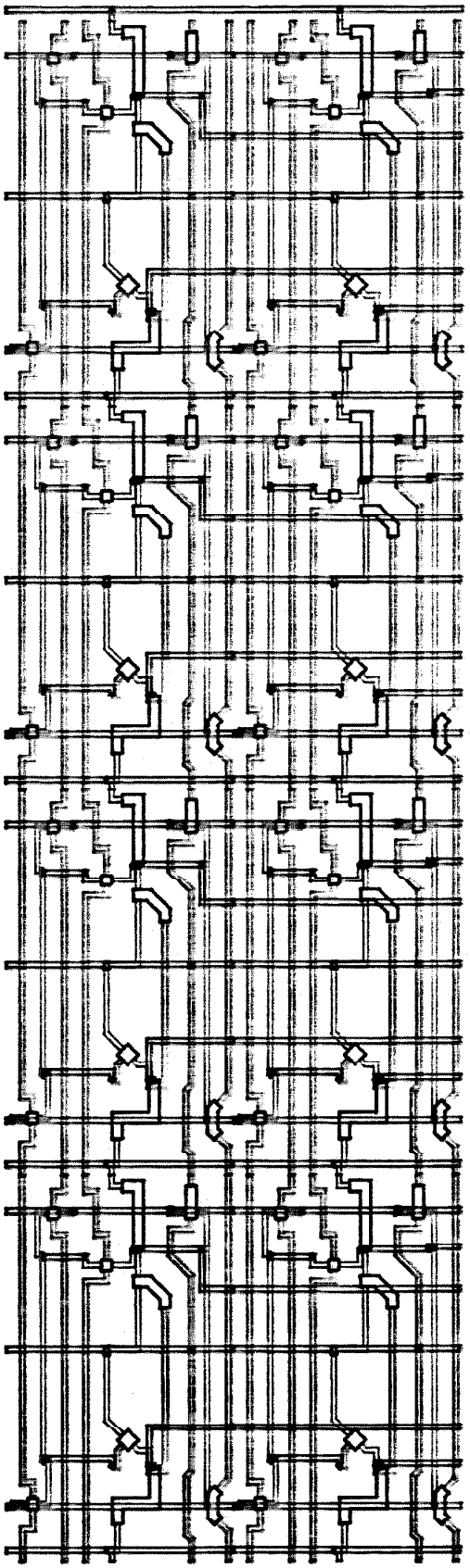


FIGURE 6-13. 4 BIT WIDE MEMORY PATH.

6.3.2 MASK MAKING

To interface to pattern generation software for mask making the system produces CIF 2.0 [Sproull80] descriptions of the design at the request of the designer. Suitable comments are inserted into the file, compatible with extensions already proposed by other researchers [Hon79]. CIF code files tend to become long with large designs and therefore the shift register cell was used to illustrate this subsystem (Figure 6-14).

```
DS    1;
(SRCELL.A);
L ND;
P  600,500 600,200 400,200 400,500;
L NM;
P  500,400 2100,400 2100,0 500,0;
L ND;
P  200,900 800,900 800,300 200,300;
L NP;
P  100,700 900,700 900,500 100,500;
P  800,700 1000,700 1000,500 800,500;
L ND;
P  600,1100 600,700 400,700 400,1100;
L NM;
P  500,2300 2100,2300 2100,1900 500,1900;
L ND;
P  1600,1300 1600,1100 1000,1100 1000,1300;
L NP;
P  1400,1400 1400,1000 1200,1000 1200,1400;
L NM;
P  0,400 500,400 500,0 0,0;
L NP;
P  0,700 200,700 200,500 0,500;
P  1900,1000 1900,700 2100,700 2100,500 1700,500 1700,1000;
L ND;
P  600,1100 600,1000 900,1000 900,1300 1200,1300 1200,1100
 1100,1100 1100,800 400,800 400,1100;
L NP;
L NM;
P  0,2300 500,2300 500,1900 0,1900;
L ND;
P  400,1100 400,1800 600,1800 600,1100;
L NP;
P  200,1100 200,1800 800,1800 800,1100;
L NI;
P  250,950 250,1950 750,1950 750,950;
L NP;
P  1400,2600 1400,1300 1200,1300 1200,2600;
L ND;
P  1400,1300 1900,1300 1900,1000 1700,1000 1700,1100 1400,1100;
L NP;
P  1400,1100 1400,0 1200,0 1200,1100;
L ND;
P  600,2100 600,1800 400,1800 400,2100;
L NM;
B    600   400 500,1100 0,1;
L NC;
B    400   200 500,1100 0,1;
L ND;
B    400   400 500,1000;
L NP;
B    300   400 500,1250 0,1;
L NM;
B    400   400 500,200;
L NC;
B    200   200 500,200;
L ND;
B    400   400 500,200;
L NM;
B    600   400 1800,1000 0,-1;
L NC;
B    400   200 1800,1000 0,-1;
L ND;
B    400   400 1800,1100;
L NP;
B    300   400 1800,850 0,-1;
L NM;
B    400   400 500,2100;
L NC;
B    200   200 500,2100;
L ND;
B    400   400 500,2100;
L ND;
B    200   200 500,1100;
DF;
E
```

FIGURE 6-14. SHIFT REGISTER CELL - CIF CODE.

6.3.3 VERIFICATION

For reasons previously explained it was decided to use the shift register design example to demonstrate the verification subsystems. The logic diagram, stick diagram and artwork for the basic cell are shown in Figure 6-1.

6.3.3.1 DIMENSIONAL DESIGN RULE CHECKING

A complete design rule checker (DRC) was beyond the scope of this thesis. A few illustrative examples were programmed to demonstrate the feasibility of an integrated subsystem of this type.

Existing DRC systems spend a substantial proportion of their time checking the widths of wires and transistors. This is unnecessary under this design discipline because minimum widths are already set and there is no way of specifying arbitrary boxes or polygons. Similarly metal, polysilicon and diffusion overlap around contact holes and polysilicon and diffusion extensions around transistors are automatically produced. Both of these gains are due to the principle of correctness by construction.

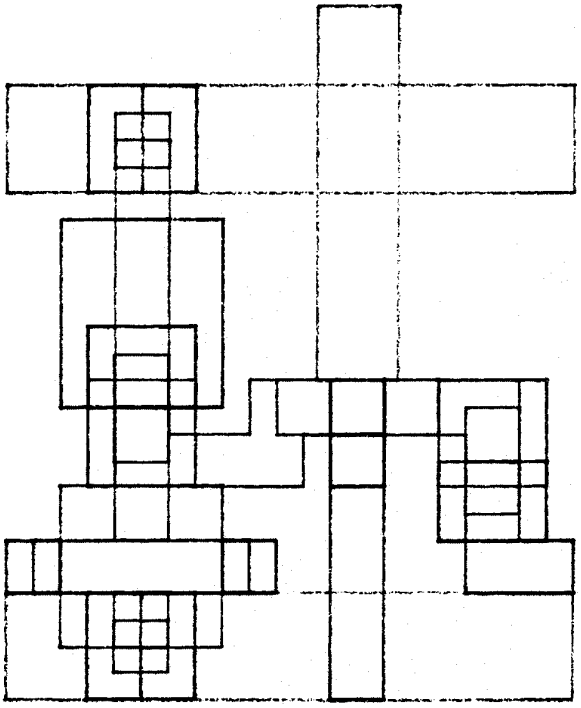
Separation checking is the major remaining problem. Within this class are two similar checks; separation between geometries on the same layer and separation between geometries on different layers. For an example of the first type consider metal to metal separation. A specially constructed incorrect design is used as a test example. The artwork and stick diagram for this example are shown in Figures 6-15(a) & 6-15(b). The algorithm is as follows:

(i) Extract metal geometry (Figure 6-15(c))

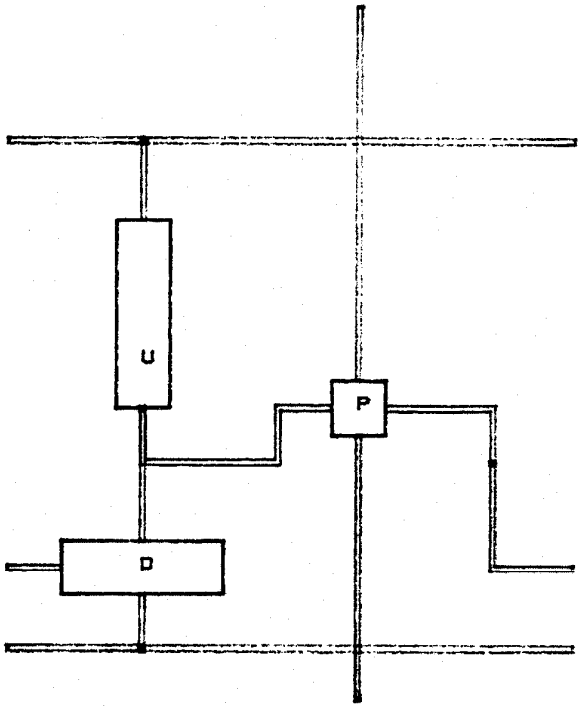
(ii) Inflate the shapes by half the minimum separation of 3 lambda (Figure 6-15(d))

(iii) Find the intersection of the merged shapes (Figure 6-15(e))

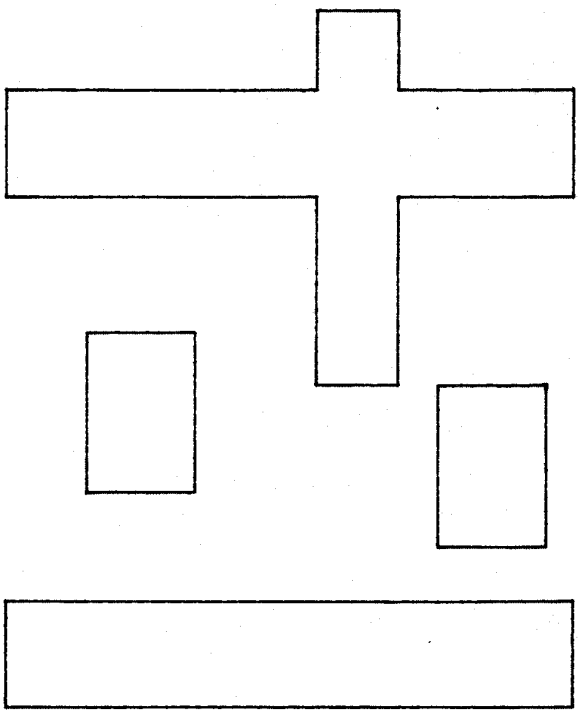
(iv) If this set is empty then the design contains no violations, otherwise the output of Figure 6-15(e) can be fed back to the designer superimposed on the original artwork to show the position of the violations (Figure 6-15(f) & 6-15(g)).



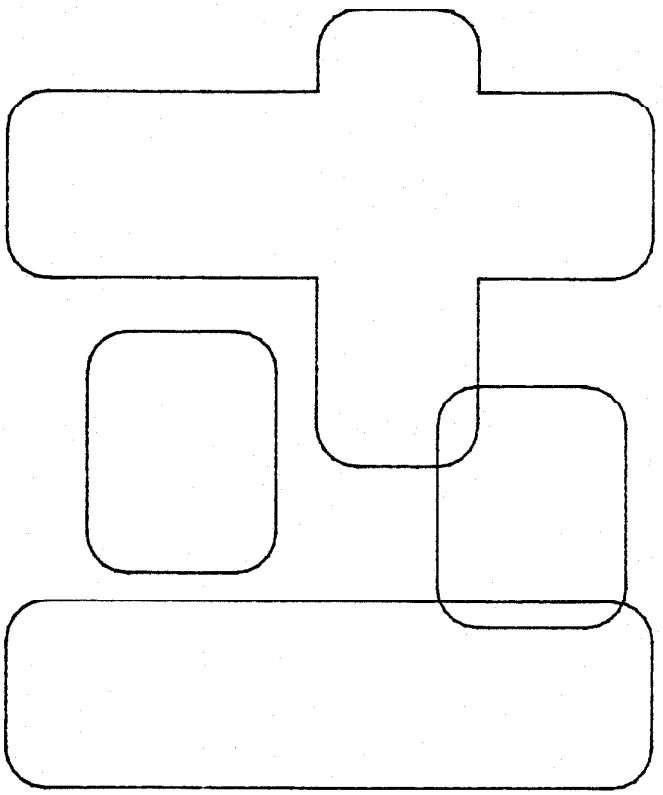
(A) LAYOUT



(B) STICK DIAGRAM

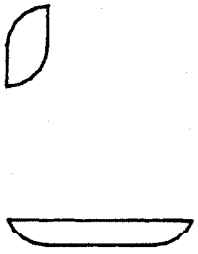


(C) EXTRACT METAL

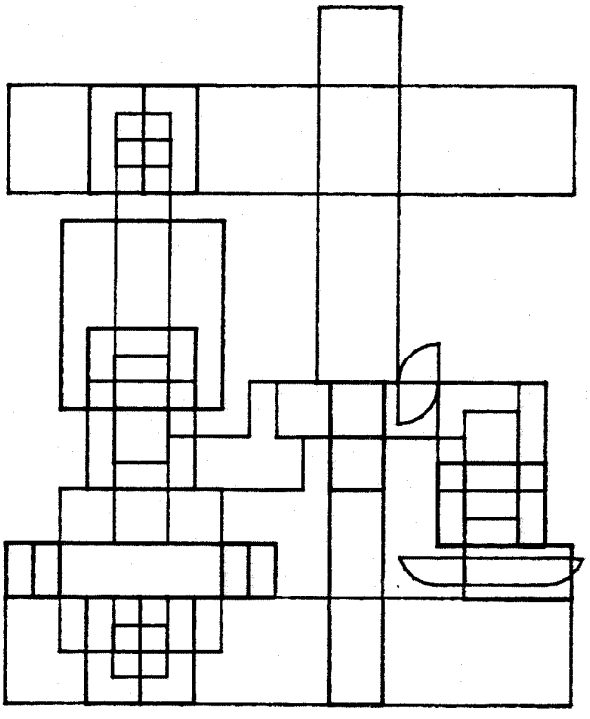


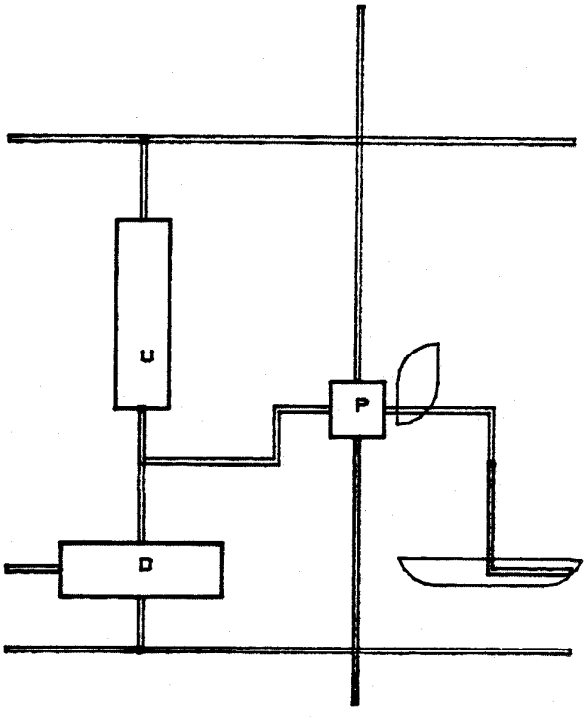
(D) INFLATE METAL

FIGURE 6-15. DRC METAL-METAL SEPARATION.



(E) INTERSECTION AREAS





(F) SUPERIMPOSE LAYOUT (G) SUPERIMPOSE STICKS

FIGURE 6-15. DRC METAL-METAL SEPARATION.

The transistor to contact hole separation was chosen to illustrate the checking of a rule which defines the minimum separation between two different layers. Using the special cell which has been constructed to contain a violation. The artwork and stick diagram for this example are shown in Figures 6-16(a) & 6-16(b). The algorithm is as follows:

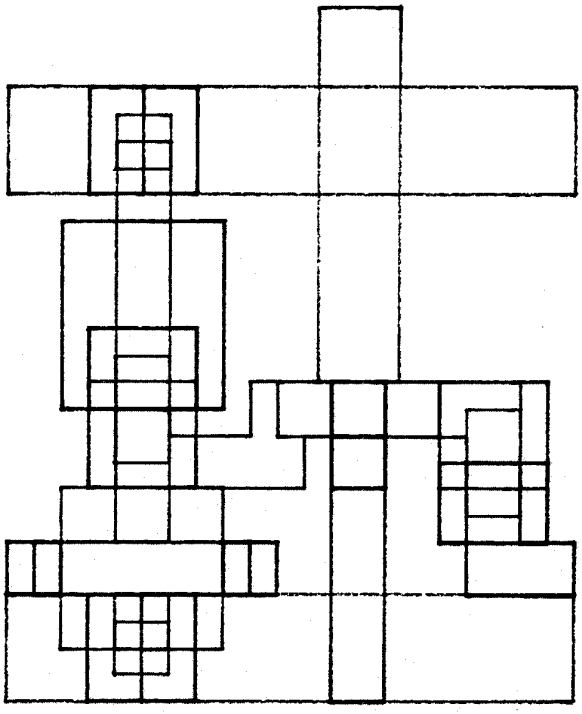
(i) Extract all transistor geometry (Figure 6-16(c)).

(ii) Extract the geometry of all non-butting contacts (Figure 6-16(d)).

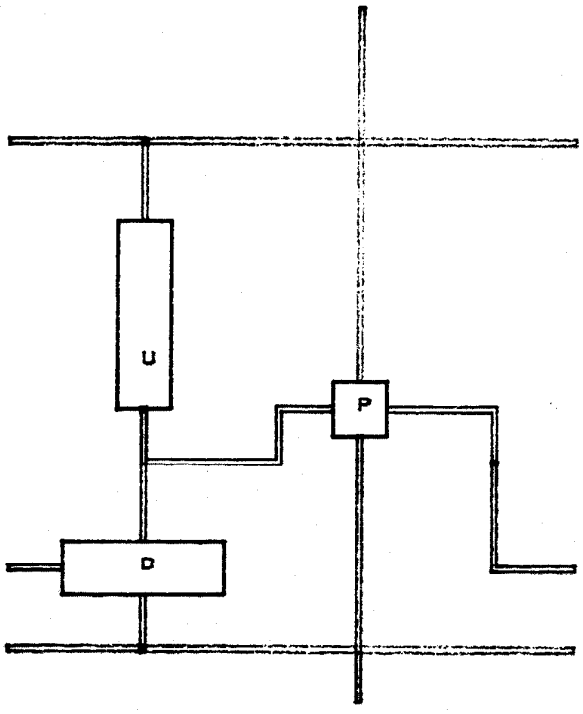
(iii) Inflate both sets separately (Figure 6-16(e) & 6-16(f)).

(iv) Find the intersection of the two sets (Figure 6-16(g)).

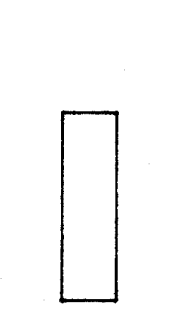
(v) Report the violations as previously outlined (Figure 6-16(h) & 6-16(i)).



(A) LAYOUT

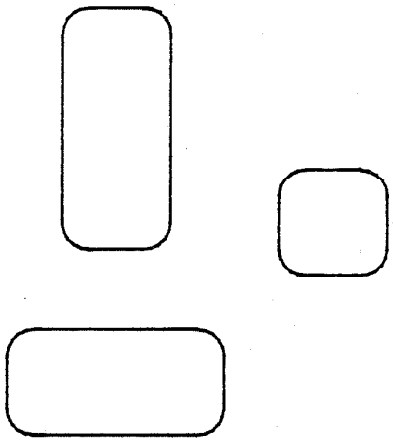


(B) STICK DIAGRAM



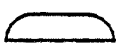
(C) EXTRACT TRANSISTORS (D) EXTRACT CONTACTS

FIGURE 8-16. DRC TRANSISTOR-CONTACT SEPARATION.



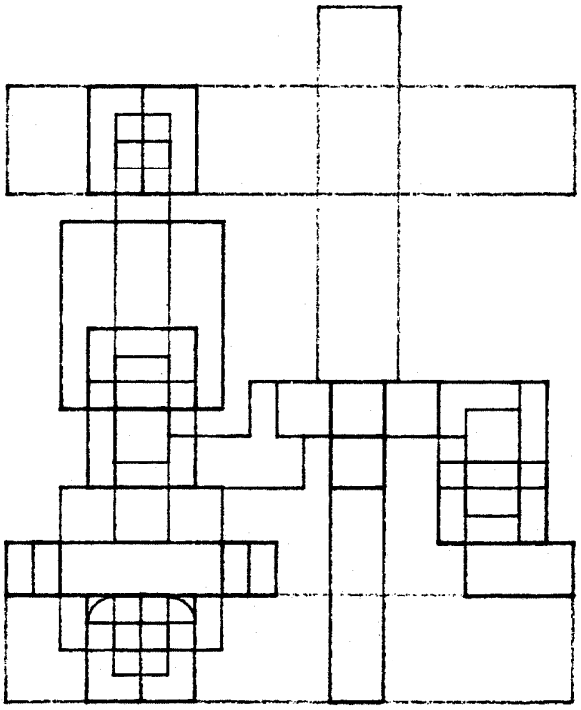
(E) INFLATE TRANSISTORS

(F) INFLATE CONTACTS

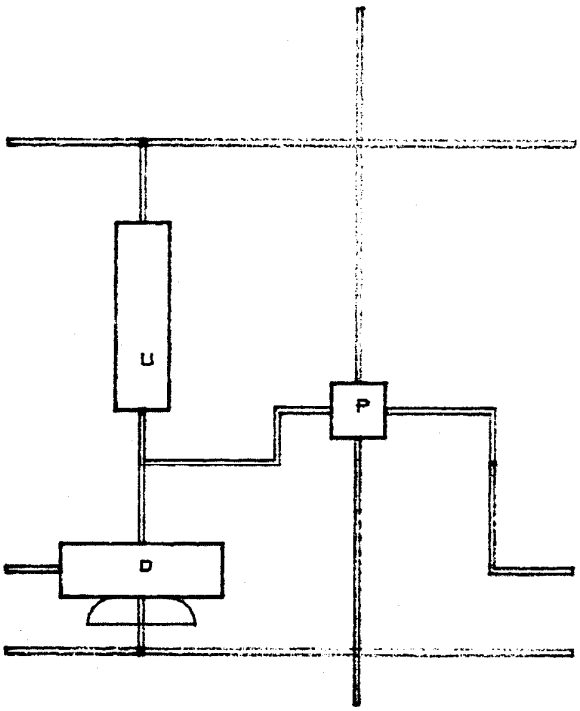


(G) INTERSECTION AREA

FIGURE 8-18. DRC TRANSISTOR-CONTACT SEPARATION.



(H) SUPERIMPOSE LAYOUT



(I) SUPERIMPOSE STICKS

FIGURE 6-16. DRC TRANSISTOR-CONTACT SEPARATION.

6.3.3.2 ELECTRICAL CONSIDERATIONS

IC designers must calculate a number of electrical characteristics of circuits. These include resistance, capacitance, power density and pullup/pulldown ratios. This design system can provide the designer with electrical values corresponding to the models of the process which the design environment contains. Through the names of transistors and coordinodes that have been used by the designer in describing the circuit the system can interface to the designer and provide visual feedback.

Once again, a complete array of calculations has not been provided but instead a subset of illustrative examples has been chosen. Suppose the designer wishes to check pullup/pulldown ratios for the shift register cell. Then a terminal session might be as shown in Figure 6-17(a). Since the connectivity of the circuit is known it would be possible for the system to discover which pulldowns were associated with a specified pullup but this facility has not been included.

Another example using the same cell is the calculation of the resistance of the clocked control line running vertically through the block. A terminal session (Figure 6-17(b)) illustrates this example.

```
command:electrics
calculate:help
ratios - calculate pullup/pulldown ratios
resistance - calculate selected item resistance
help - print this information
return - go back to command level
calculate:ratios
pullup:pull
* to end
pulldown:inv
ratio=   10.5000
pulldown:*
```

FIGURE 8-17(A). ELECTRICAL CALCULATIONS - RATIOS.

```
calculate:resistance
end with *
end coordinode 1:gx
end coordinode 2:gout
resistance=    0.1200
end coordinode 1:ps.out
end coordinode 2:pout
wire not found
end coordinode 1:ps.out
end coordinode 2:phout
resistance=  165.0000
end coordinode 1:*
calculate:return
command:
```

FIGURE 8-17(B). ELECTRICAL CALCULATIONS - RESISTANCES.

6.3.3.3 CONNECTIVITY

Connections intended by the designer must exist physically due to modelling in the system. Because the system maintains such a comprehensive description of the design it can detect connectivity problems during design construction. For example, in constructing the erroneous design for the remainder of this section the terminal feedback (Figure 6-18) shows that the system found a possible missing contact (due to the wire from "phin" to "ps.in" being on the metal instead of the polysilicon layer).

```
          help
srcarray - construct an array of shift register cells
srcell - construct a shift register cell definition
help - print this information
return - return to main command level
definition:srcell
name version:a
change defaults (y or n):y
gnd,sig,vdd,phi,xlim,ylim (2,6,21,13,21,26):2 5 21 13 21 26
contact violation at PS.IN
SRCELL.A
definition:
```

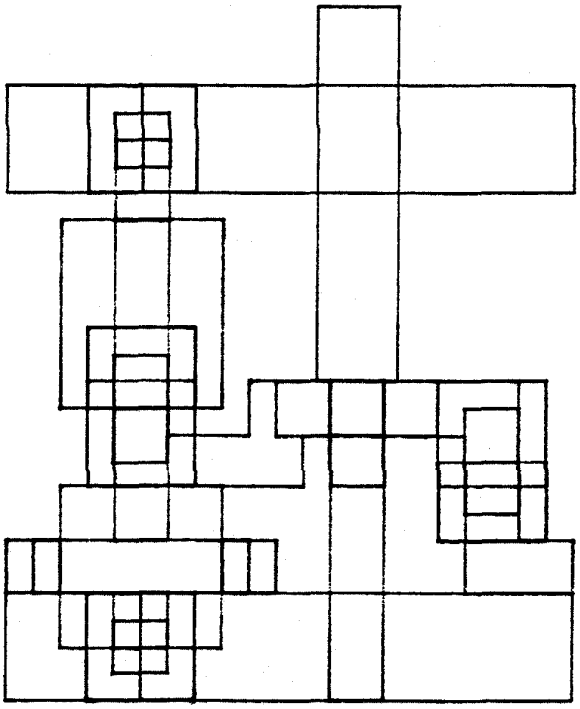
FIGURE 6-18. CONNECTIVITY - TERMINAL SESSION

It is essential to detect unintentional physical connections. As an example, consider the metal layer and search for connections which exist in the layout but which have not been specified by the designer. A specially constructed incorrect design is used as an example. The artwork and stick diagram for this example are shown in Figures 6-19(a) & 6-19(b).

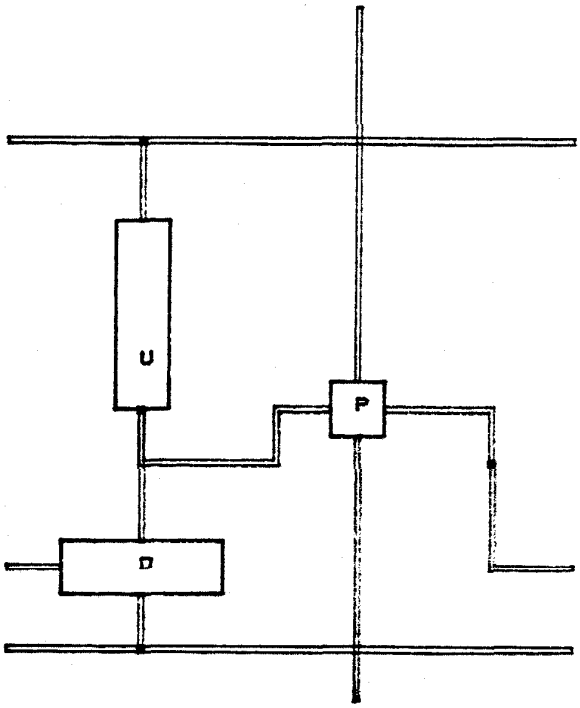
(i) For each electrical node extract the metal geometry (Figure 6-19(c) & 6-19(d))

(ii) Find the intersection of the electrical nodes (Figure 6-19(e))

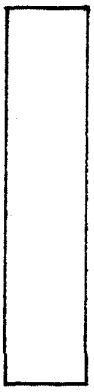
(iii) If this set is empty then the design contains no erroneous connections on the metal layer, otherwise the output of Figure 6-19(e) can be returned to the designer to show the position of the errors (Figure 6-19(f) & 6-19(g)).



(A) LAYOUT



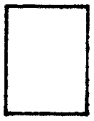
(B) STICK DIAGRAM



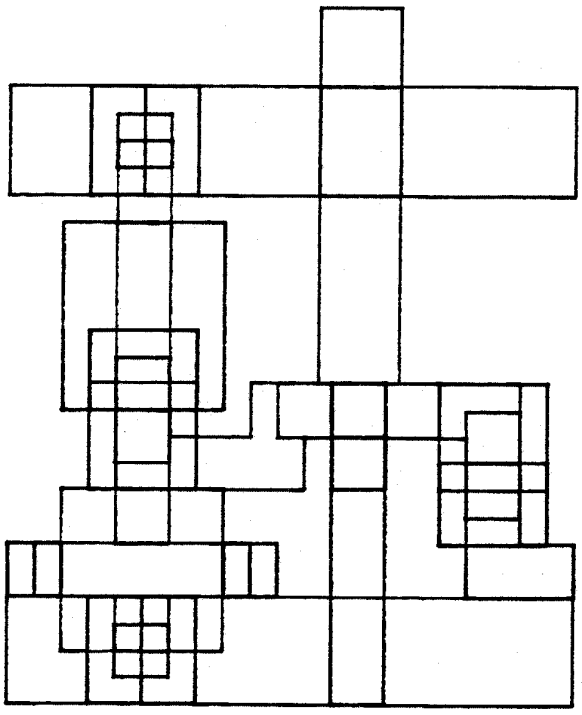
(C) VERIFIED METAL

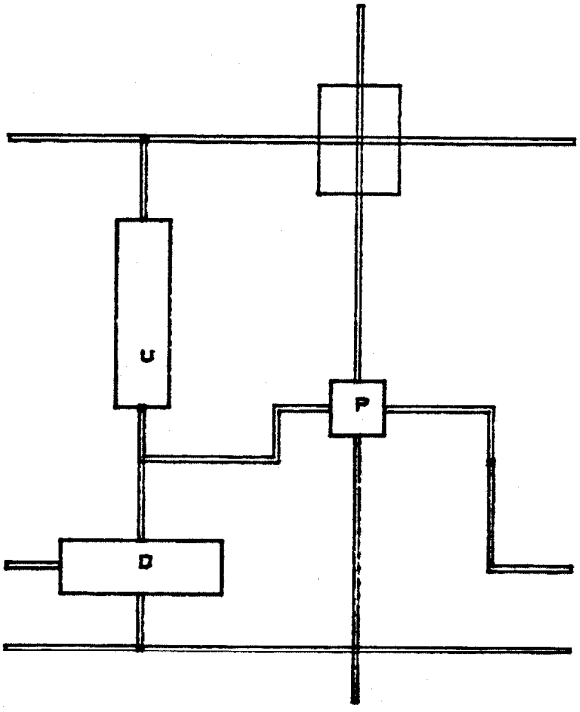
(D) NEXT TO CHECK

FIGURE 8-18. CONNECTIVITY CHECKING.



(E) INTERSECTION AREA





(F) SUPERIMPOSE LAYOUT (G) SUPERIMPOSE STICKS

FIGURE 8-18. CONNECTIVITY CHECKING.

6.3.3.4 SIMULATION

It is very important that simulation programs work on data that matches the design artwork that is going to be used for mask making. Designing artwork and independently entering input to a simulation program allows discrepancies to develop between the two forms. Therefore simulators must either be integrated within the design system or must operate on data files produced by design systems which contain the relevant information. This is the philosophy adopted by this design system and both operational alternatives are employed over the variations of simulators available in the system.

6.3.3.4.1 CIRCUIT LEVEL

Circuit level simulation is a specialist art that has spawned a number of simulation programs more or less applicable to MOS design. SPICE [Nagel75] has become an industry standard and that factor, combined with its accessibility combined to make it the obvious choice for interfacing to this system. Therefore, using the component and connectivity information contained in the design description and the modelling information included in the design environment for NMOS, the designer can automatically produce a SPICE input file. The designer is provided with a user interface internal to the system

through which to identify the power and ground nodes, specify the input transitions and define the output formats. A typical terminal session is shown in Figure 6-20(a). The SPICE input file describing the shift register cell and the associated simulator output are shown in Figures 6-20(b) & 6-20(c) respectively.

```
command:spice
file name:srcell
identify vdd coordinode:vin
identify gnd coordinode:gin
identify outputs (end with *):sigout
identify outputs (end with *):*
all times in nanoseconds
identify inputs (end with *):sigin
time, voltage pair (-1 to end)
pair:0 5
pair:45 5
pair:50 0
pair:100 0
pair:-1
identify inputs (end with *):phin
time, voltage pair (-1 to end)
pair:0 0
pair:10 0
pair:15 5
pair:25 5
pair:30 0
pair:60 0
pair:65 5
pair:75 5
pair:80 0
pair:-1
identify inputs (end with *):*
output time step and period:1 100
print node (* to end):*
plot node (* to end):sigin
plot node (* to end):phin
plot node (* to end):sigout
plot node (* to end):*
```

FIGURE 6-20(A). SPICE TERMINAL SESSION.

```
* LAMBDA=    3U
*      2 : BULK
*      1  : INV.DRN - BUFFX - PULL.SRC - PULL.IN - PULL.OUT - PS.SRC
*      0  : INV.SRC - GX - GIN - GOUT
*      3  : INV.OUT - INV.IN - SIGIN
*      4  : PULL.DRN - VX - VIN - VOUT
*      5  : PSX - PS.DRN - SIGOUT
*      6  : PHIN - PS.IN - PS.OUT - PHOUT
MINV      1    3    0    2  MENH  W=    18.0000U  L=     6.0000U
MPS       5    6    1    2  MENH  W=     6.0000U  L=     6.0000U
MPULL     4    1    1    2  MDEP  W=     6.0000U  L=    21.0000U
VDD    4  0 5
VB     2   0 -3
CSIG    5 0 0.015PF
VIN3    3    2  PWL (  0N  5V 45N  5V 50N  0V100N  0V)
VIN6    6    2  PWL (  0N  0V 10N  0V 15N  5V 25N  5V 30N  0V 60N  0V
+65N  5V 75N  5V 80N  0V)
.TRAN 1.000N 100N
.PRINT TRAN
.PLOT TRAN  V(  5)  V(  3)  V(  6)  (-1,5)
.MODEL MENH NMOS (LEVEL=2 TOX=7E-8 NSUB=3.5E14
+NSS=-1.882E11 XJ=.75U LD=.8 NGATE=1E23 GAMMA=.43
+NFS=1E11
+UO=825 UCRIT=6E4 UEXP=.25 UTRA=.5
+CBD=12.33E-5 CBS=12.33E-5 JS=2E-5)
.MODEL MDEP NMOS (LEVEL=2 TOX=7E-8 NSUB=3.5E14
+NSS=7.78E11 NGATE=1E23 XJ=.75U LD=.8
+UCRIT=6E4 LAMBDA=0.001 UEXP=0.25 UTRA=0.5
+UO=825 CBD=12.33E-5 CBS=12.33E-5 JS=2E-5)
.END
```

FIGURE 6-20(B). SPICE INPUT FILE.

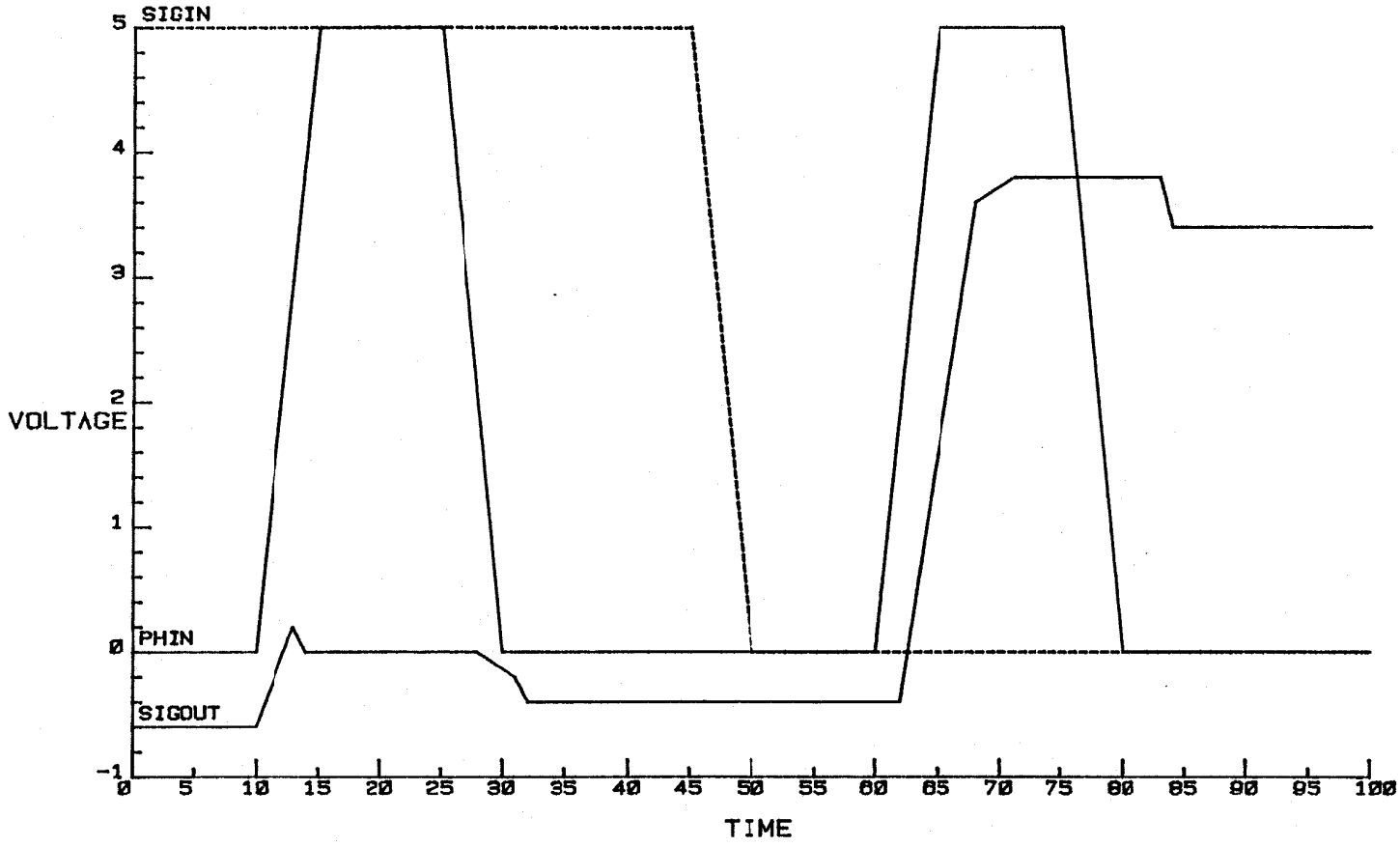
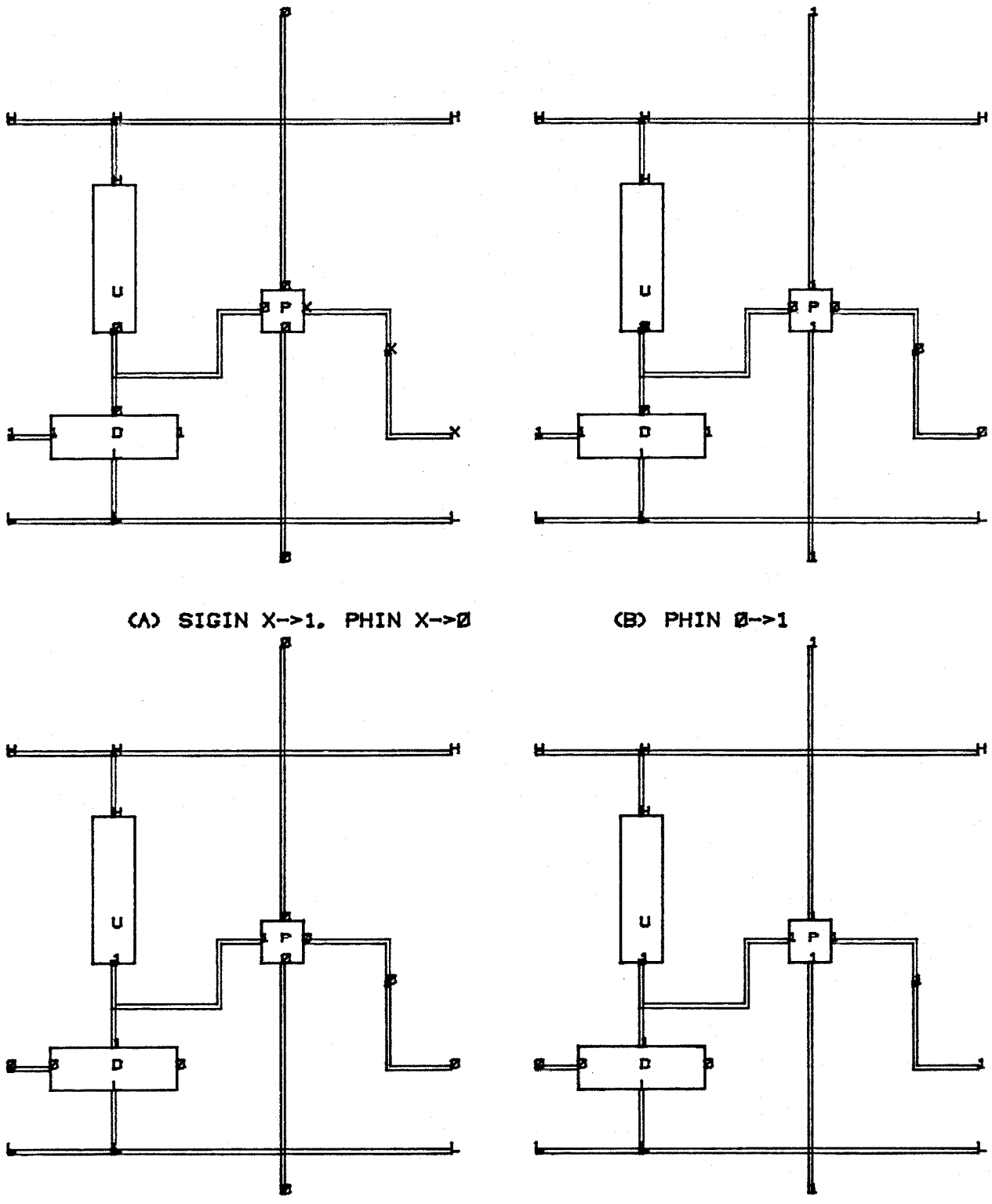


FIGURE 6-20(C). GRAPH OF SPICE OUTPUT.

6.3.3.4.2 LOGIC LEVEL

The logic level simulator is integrated within the design system and provides graphical feedback to the designer via the colour display. Textual output is also available. Input values are defined by the designer and progress through the simulation is under the control of the designer so that states may be examined at leisure. A series of snap-shots from a simulation session is shown in Figure 6-21. To simulate the effect of joining together block instances it is necessary to deal with the characteristics of hierarchical design; in particular different logic values may be held by the same net in different block instances. This causes a certain amount of unavoidable data explosion but does not require a complete enumeration of the hierarchy. Only the behavioural information need be duplicated since each block instance uses the same structural description. Figure 6-22 shows a series of snap-shots from a simulation of a register cell array. Figure 6-23 illustrates an alternative mode of display in which all values down through the levels of abstraction are shown.



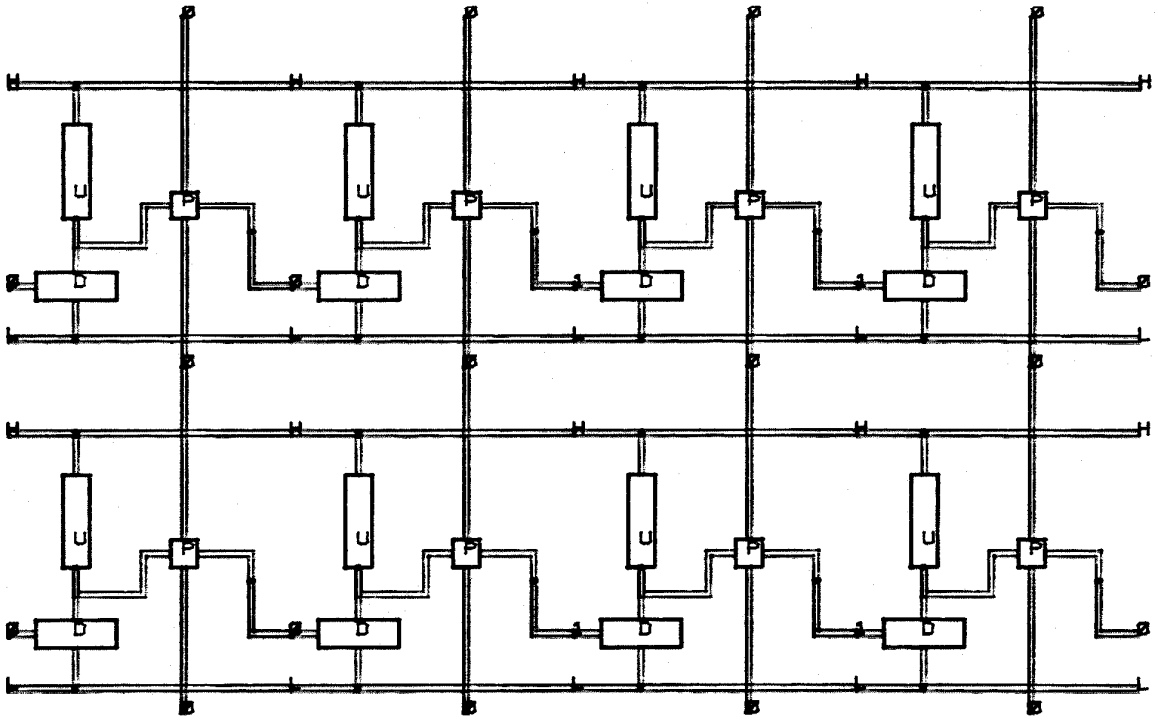
(A) SIGIN X->1, PHIN X->Ø

(B) PHIN Ø->1

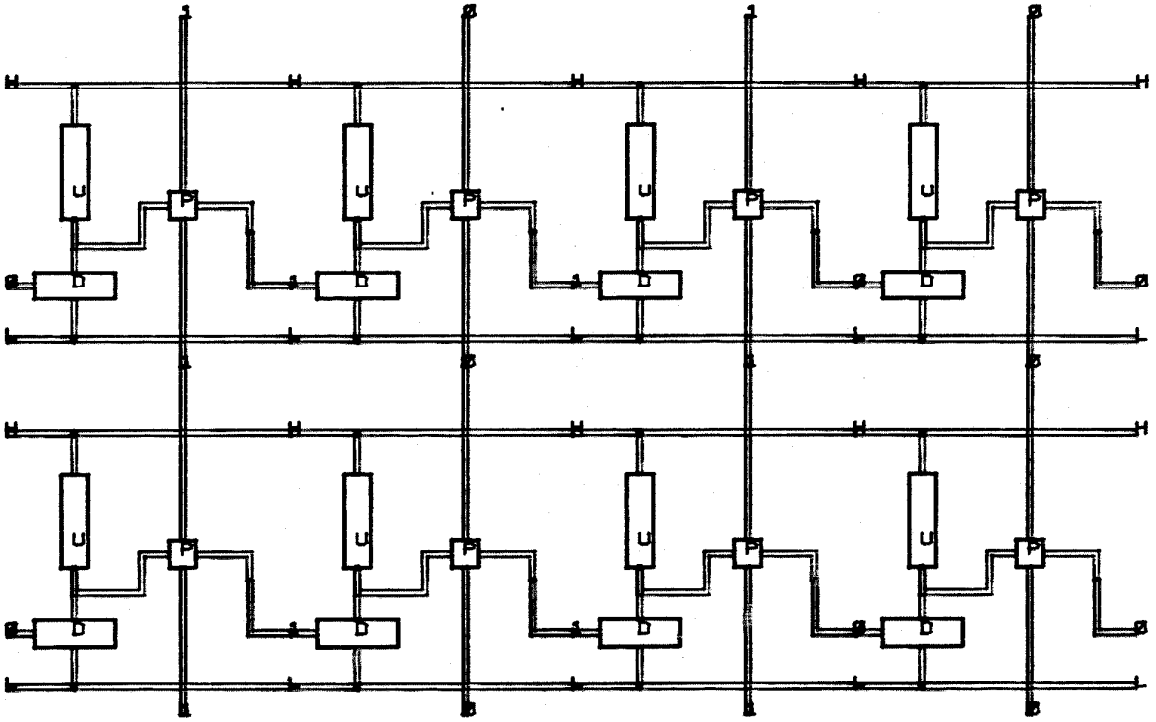
(C) SIGIN 1->Ø, PHIN 1->Ø

(D) PHIN Ø->1

FIGURE 6-21. SHIFT REGISTER CELL - LOGIC SIMULATION.

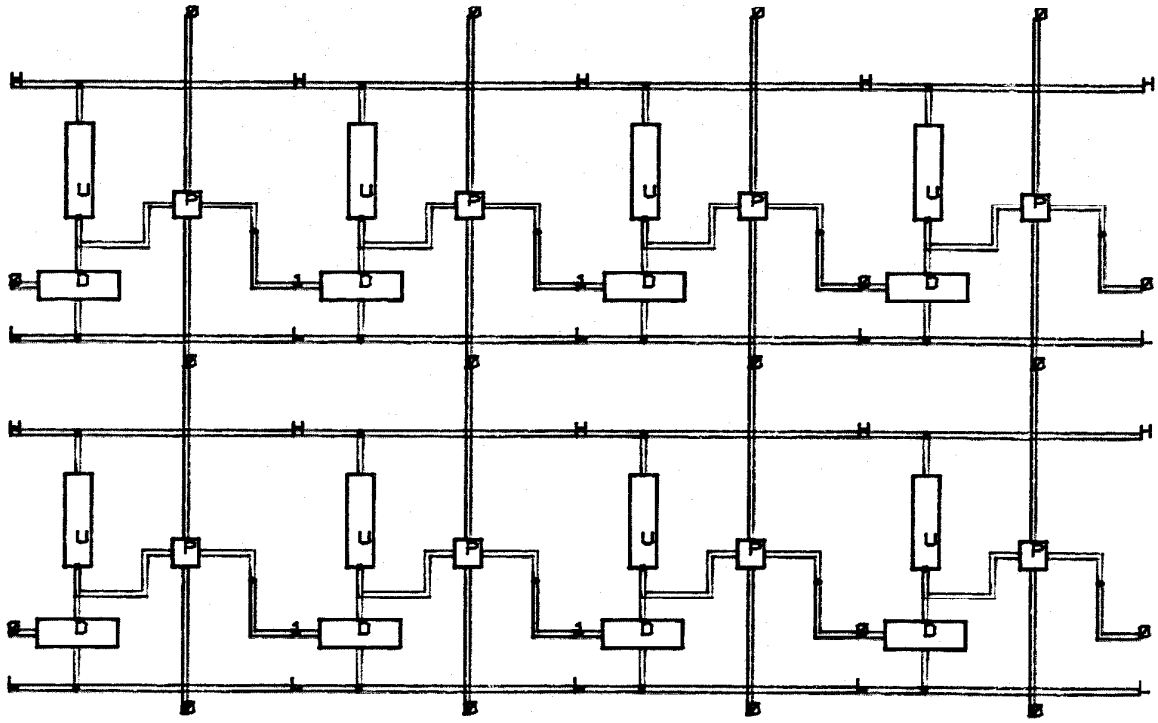


(A) SIGIN 1->Ø. PHI2 1->Ø

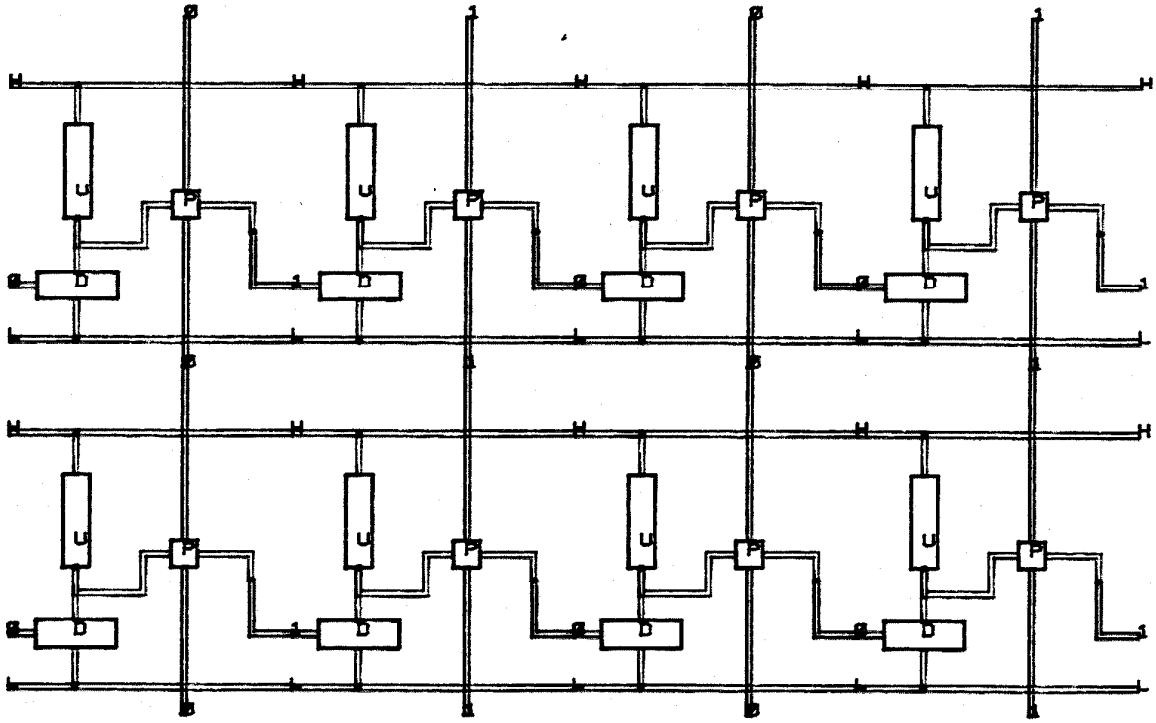


(B) PHI1 Ø->1

FIGURE 6-22. SHIFT REGISTER ARRAY - LOGIC SIMULATION.



(C) PHI1 1->Ø



(D) PHI2 Ø->1

FIGURE 6-22. SHIFT REGISTER ARRAY - LOGIC SIMULATION.

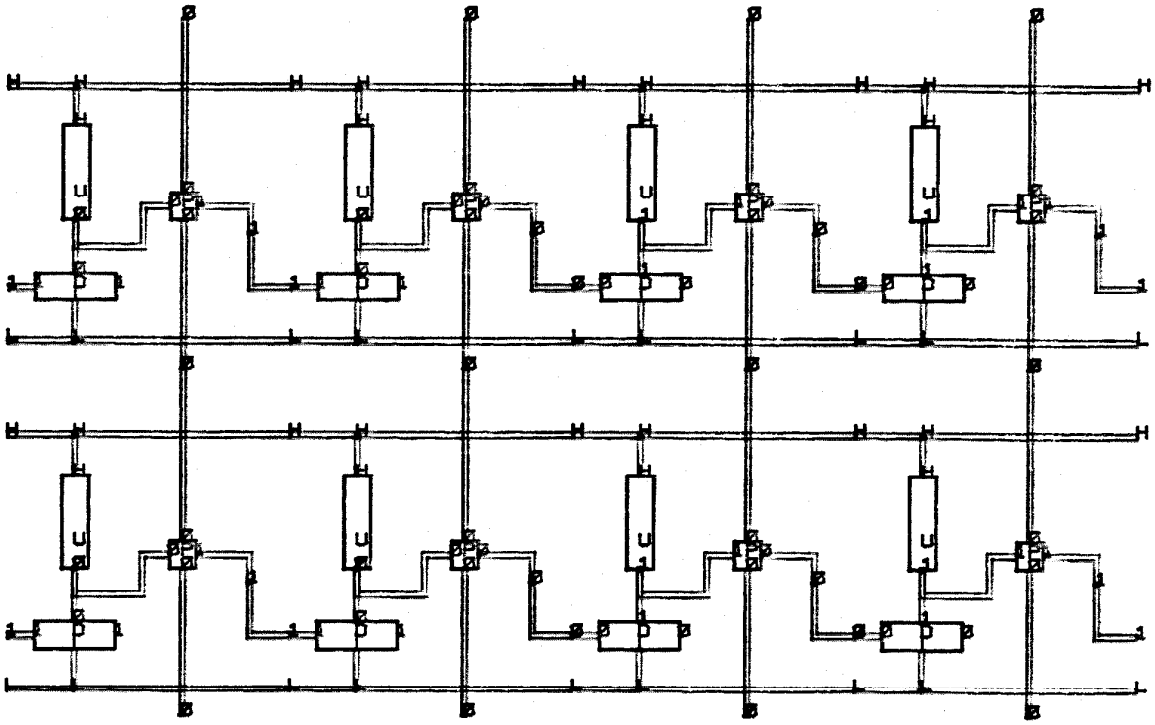


FIGURE 6-23. SHIFT REGISTER ARRAY - FULL NODE DISPLAY.

6.3.3.4.3 BLOCK LEVEL

When describing the behaviour of a design in terms of its blocks it may be desirable to use a procedure. This allows more concise definition of the design's function and more efficient simulation. This design system caters for such behavioural descriptions via the SIMULA procedural attribute facility. A procedure describing the behaviour of the shift register is included in Appendix A. Output is exactly the same as Figure 6-22.

Consistency between the higher and lower levels of behaviour description is obviously desirable but proof is difficult to obtain. Enumerating and comparing all possible sequences of inputs to different simulation level descriptions is impractical. Some degree of assurance might be given by a limited comparison and the system could be made to support a suitable designer directed session.

7. CONCLUSIONS

This thesis has shown that the provision of models with components in all three descriptive domains i.e. the structural, physical and behavioural, allows the designer to construct a complete and consistent design description. Through this methodology the design system can provide a high level of functionality and achieve reliable verification.

In comparison with traditional design systems the approach adopted here has the major benefit of reliable verification and the capability to generate a number of representations of the design at different levels of abstraction. Block diagrams, stick diagrams, check plots, and data for mask making and the full spectrum of verification procedures can all be generated from or cause operations on the same data structure which has been constructed using procedural models of the design components.

The implementation of such a design system is centred around a high level, general purpose programming language environment. An object oriented language, such as SIMULA, provides the most suitable programming base for such a system. Models of both primitive components and composite blocks are provided by the system. Flexibility and

extensibility are assured by its inherent programmability.

Increased functionality is achieved at the cost of more rigorous and time consuming design input to the system and increased computing resource utilisation during the execution phase. However, it is not possible to discuss performance in relation to other design systems as no comparable system exists at this time.

7.1 IMPROVEMENTS

Evaluation of the system from a user standpoint suggests a number of possible improvements. The most significant of these is the division of the design task into two parts viz cell design and chip assembly. These tasks are not completely separable but have sufficiently different requirements that two complementary systems are needed to service them.

Cell design deals with primitive components being connected in the most dense formation a designer can devise. The most relevant representation a design system can utilise is graphical. Artwork and stick diagrams provide the structural and physical forms.

Therefore the designer requires a design system with two coordinated modules. The cell design system should be

capable of editing a design in either graphical or textual form, interchangeably and consistently [Trimberger79]. The textual form then becomes the high level language input to the chip integration system. This implies that the cell design system must be able to interpret at least a subset of the high level language used for the design description.

Chip integration involves instantiating cells to build more complex blocks in a hierarchical schema. A language description such as that discussed in this thesis provides the necessary parameterisation and powerful control structures that this phase of design requires. Guaranteeing that the parameterisation of the description is correct has to depend partly on the designer's conception of the system architecture and is therefore not an area of design verification in which the system can provide complete assurance. Block level simulation can be used by the designer to check that the system behaves as expected at the highest level of abstraction. Successively lower levels of the hierarchy could be simulated and the results compared with those obtained at the level above. There is no automatic procedure for this at present in the system. It would be possible to provide such a facility by arranging for a duplicate behavioural state structure to be built, simulating both incarnations concurrently under different levels of behavioural

description and comparing the results.

System performance is another area of concern. In order to achieve acceptably low response times to most operations and to minimise memory utilistaion it may be desirable to develop more dynamic data structures particularised towards single functions rather than the all-embracing static data structure currently in use in the system.

7.2 LIMITATIONS

A design system of the type described in this thesis has a number of operational drawbacks which are a result of its greater functionality. To get more function out the IC design system writer and the IC designer have to put more effort in i.e. producing and using a system which deals in structural, physical and behavioural descriptions requires more work than producing and using a traditional geometrically based system. However, the advantages of this approach, as summarised at the beginning of this chapter, are thought to be a worthwhile gain.

7.2.1 TECHNOLOGY DEPENDENCE

The primitive models described in Chapter 4 are specific to NMOS technology. Other technologies would

require different primitive models although they would follow broadly similar lines. However, the major part of the programming environment, the approach to design and the description of block structure would remain the same.

7.2.2 LIBRARIES

The design system performs most efficiently given a library of parameterised blocks specific to a particular systems architecture e.g. the OM2 ALU data path [Mead80]. A number of designers can then create a number of designs of the same general class. New floor plans could be generated from the same basic cells. However, radical changes in the character of the target systems architecture would require the redesign of the block library.

7.2.3 PERFORMANCE

The structured IC design methodology discussed in Chapter 4 and integrated within the IC design environment described in Chapter 5 is intended to facilitate the layout of identically pitched blocks. In effect this means that cells will be stretched to fit the cell with the largest dimension in the connect path. Although this constraint maintains regular interconnect it may cause designs to be generated that are less dense than they

could be. Comparisons of the effect on density of different design methodologies are not widely available but some initial results suggest that by avoiding random interconnect regular designs regain globally the density they may lose locally [Heller79, Tarolli79].

7.3 SUMMARY

The relevance of this thesis is not diminished by details of implementation or performance; the philosophy remains the same. Structured design of VLSI requires complete, consistent descriptions to be captured by the computer aided design system in order to provide all the facilities a designer cannot do without in this progressively more complex field.

GLOSSARY

ALU	Arithmetic Logic Unit
Block	Design module composed of design primitives and lower level modules
Cell	Design module cf block
Diffusion	Doped conducting region of an IC
DRC	Design Rule Checking
ECL	Emitter Coupled Logic
IC	Integrated Circuit
ISP	Instruction Set Processor
LSI	Large Scale Integration
Metal	Aluminium conducting layer of an IC
MOS	Metal Oxide Semiconductor
NMOS	N-channel MOS

PCB	Printed Circuit Board
Pixel	Minimum size picture element
PLA	Programmable Logic Array
PMS	Processor Memory Switch
Polysilicon	Polycrystalline silicon conducting layer of an IC
RAM	Random Access Memory
ROM	Read Only Memory
RT	Register Transfer
Shift register	Array of cells each storing data during one time cycle
SOS	Silicon on Sapphire
SSI	Small Scale Integration
ULA	Uncommitted Logic Array
VLSI	Very Large Scale Integration

REFERENCES

Some references cite internal memos from the Department of Computer Science, California Institute of Technology. These are unpublished and should be regarded as private communications.

[Aho74] Aho A. V., Hopcroft J. E. & Ullman J. D.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, 1974.

[Ayres79] Ayres R.
Silicon Compilation - A Hierarchical Use of PLA's.
Caltech Conference on VLSI, January 1979.

[Baird75] Baird H. S. & Cho Y. E.
An Artwork Design Verification System.
Proceedings of the 12th Design Automation Conference 1975.

[Baird77] Baird H. S.
Fast Algorithms for LSI Artwork Analysis.
Proceedings of the 14th Design Automation Conference, 1977.

[Barbacci77] Barbacci M. R., Barnes G. E., Cattell R. C. & Siewiorek D. P.
The ISPS Computer Description Language.
Technical Report, Department of Computer Science, Carnegie-Mellon University, 1977.

[Barton80] Barton E. E. & Buchanan I.
The Polygon Package.
CAD Journal, January 1980.

[Bell71] Bell C.G. & Newell A.
Computer Structures: Readings and Examples.
McGraw-Hill, 1971.

[Bell72] Bell C. G., Grason J. & Newell A.
Designing Computers and Digital Systems.
Digital Equipment Corporation, 1972.

[Bell78] Bell C. G., Mudge J. C. & McNamara J. E.
Computer Engineering.
Digital Equipment Corporation, 1978.

[Blakeslee79] Blakeslee T. R.
Digital Design with Standard MSI & LSI.
Wiley-Interscience, Second Edition, 1979.

[Birtwist73] Birtwistle G.M., Dahl O-J., Myhrhaug B. &

Nygaard K.
SIMULA begin
Auerbach Publishers Inc., 1973.

[Buchanan78] Buchanan I.
A Note on Design Rule Checking.
Caltech SSP File #1917, 1978.

[Buchanan79] Buchanan I. & Gray J. P.
Models for Structured IC Design.
Department of Computer Science, University of Edinburgh, CSR-48-79, 1979.

[Correia77] Correia M. & Petrini F. B.
Introduction to an LSI Test System.
Proceedings of the 14th Design Automation Conference, 1977.

[Dahl72] Dahl O-J., Dijkstra E. W. & Hoare C. A. R.
Structured Programming.
Academic Press, London, 1972.

[Dietmeyer78] Dietmeyer D. L.
Logic Design of Digital Systems.
Allyn & Bacon, 2nd Edition, 1978.

[Dobes76] Dobes I. & Byrd R.

The Automatic Recognition of Silicon Gate Transistor Geometries: An LSI Design Aid Program.
Proceedings of the 13th Design Automation Conference, 1976

[Dunlop78] Dunlop A. E.
SLIP: Symbolic Layout with Compaction.
Bell Laboratories, Murray Hill, New Jersey, 1978.

[Eades76] Eades J. D.
The Design of an Interactive Computer System for Microelectronic Mask Making.
Ph.D. Thesis, University of Edinburgh, 1976.

[Eichelberg77] Eichelberger E. B. & Williams T. W.
A Logic Design Structure for LSI Testability.
Proceedings of the 14th Design Automation Conference, 1977.

[Fairbairn78] Fairbairn D. G. & Rowson J. A.
ICARUS: An Interactive Integrated Circuit Layout Program.
Proceedings of the 15th Design Automation

Conference, 1978.

[Gibson76] Gibson D. & Nance S.
SLIC - Symbolic Layout of Integrated Circuits.
Proceedings of the 13th Design Automation Conference, 1976.

[Gray79A] Gray J. P.
A VLSI Design Philosophy and Support Software.
Caltech SSP File #3240, 1979.

[Gray79B] Gray J. P.
Structured Design Notes.
Caltech SSP File #3354, 1979.

[Guibas79] Guibas L. J., Kung H. T. & Thompson C. D.
Direct VLSI Implementation of Combinatorial Algorithms.
Caltech Conference on VLSI, January 1979.

[Heller77] Heller W. R., Mikhail W. F. & Donath W. E.
Prediction of Wiring Space Requirements for LSI.
Proceedings of the 14th Design Automation Conference, 1977.

[Heller78A] Heller W. R.
Wireability Study for Custom Chips.
Caltech SSP File #1452, 1978.

[Heller78B] Heller W. R.
Relation of Architecture and Subsystem Function to Possible Package Configurations.
Caltech SSP File #1989, 1978.

[Heller79A] Heller W. R.
An Algorithm for Chip Planning.
Caltech SSP File #2806.

[Heller79B] Heller W. R.
Wireability of Packaging for LSI and VLSI.
Caltech SSP File #3015, 1979.

[Holzmann79] Holzmann G. J.
Coordination Problems in Multi-Processor Systems.
Delft Technical University, 1979.

[Hon79] Hon R. & Sequin C.
A Guide to LSI Implementation.
XEROX PARC,1979. (Draft).

[Hsueh79] Hsueh M-Y. & Pederson D. O.
Computer Aided Layout of LSI Circuit Building Blocks.
ISCAS 1979.

[Ichbiah79] Ichbiah J. D. et al
Rationale for the design of the ADA Programming Language.
ACM SIGPLAN Notices Vol. 14, No. 6, June 1979.

[Iverson62] Iverson K. E.
A Programming Language.
John Wiley & Sons, New York, 1962.

[Kung79] Kung H. T.
Let's Design Algorithms for VLSI Systems.
Caltech Conference on VLSI, January 1979.

[Kung80] Kung H. T. & Leiserson C. E.
Algorithms for VLSI Processor Arrays.
Contained in [Mead80].

[Johannsen79] Johannsen D.
Bristle Blocks: A Silicon Compiler.
Caltech Conference on VLSI, January 1979.

[Lattin79] Lattin W.
The Problem of the 80's for Microprocessor Design.
Caltech Conference on VLSI, January 1979.

[Lewin70] Lewin D. W.
Logical Design of Switching Circuits.
Nelson, 1970.

[Leyking79] Leyking L. W.
Database Considerations for VLSI.
Caltech Conference on VLSI, January 1979.

[Lindsay76] Lindsay B. W. & Preas B. T.
Design Rule Checking and Analysis of IC Mask Designs.
Proceedings of the 13th Design Automation Conference, 1976.

[Locanthi78] Locanthi B.
LAP: A SIMULA Package for IC Layout.
Caltech Display File #1862, 1978.

[Loosemore78] Loosemore K. J.
IC Design - Misery or Magic.
National Computing Conference, 1978.

[McCaw79] McCaw C. R.
Unified Shapes Checker - A Checking Tool for VLSI.
Proceedings of the 16th Design Automation Conference, 1979.

[McGrath79] McGrath E.
A Physical Design Rule Description.
Caltech SSP File #3236, 1979.

[McWilliams78] McWilliams T. M. & Widdoes L. C.
SCALD: Structured Computer-Aided Logic Design.
Proceedings of the 15th Design Automation Conference, 1978.

[Masumoto78] Masumoto R.T.
A 16 bit Digital Multiplier.
Thesis, E.E. Dept., Caltech, 1978.

[Mead80] Mead C. A. & Conway L. A.
Introduction to VLSI Systems.
Addison-Wesley, 1980.

[Meindl77] Meindl J. D.
Microelectronic Circuit Elements.
Scientific American, September 1977.

[Minter79] Minter C.
Charles Terminal Care Package.
Caltech SSP File #3017, 1979.

[Mitchell78] Mitchell J. G., Maybury W. & Sweet R.
Mesa Language Manual.
CSL-78-1, XEROX PARC, 1978.

[Moore79] Moore G. E.
Are We Really Ready for VLSI?
Caltech Conference on VLSI, January 1979.

[Nagel75] Nagel L. W.
SPICE2: A Computer Program to Simulate
Semiconductor Circuits.
University of California at Berkeley,
College of Engineering, 1975.

[Noyce77] Noyce R. N.
Microelectronics.
Scientific American, September 1977.

[Oldham77] Oldham W. G.
The Fabrication of Microelectronic
Circuits.
Scientific American, September 1977.

[Persky76] Persky G., Deutsch D. N. & Schweikert D. G.
LTX - A System for the Directed Automatic Design of LSI Circuits.
Proceedings of the 13th Design Automation Conference, 1976.

[Preas76] Preas B. T., Lindsay B. W. & Gwyn C. W.
Automatic Circuit Analysis Based on Mask Information.
Proceedings of the 13th Design Automation Conference, 1976.

[Preas78] Preas B. T. & Gwyn C. W.
Methods for Hierarchical Automatic Layout of Custom LSI Circuit Masks.
Proceedings of the 15th Design Automation Conference, 1978.

[Rem79] Rem M.
Mathematical Aspects of VLSI Design.
Caltech Conference on VLSI, January 1979.

[Robertson77] Robertson P. S.
The IMP-77 Language.
CSR-19-77, Department of Computer Science, University of Edinburgh.

[Rose76] Rose J. E.
An MOS Uncommitted Logic Array.
Project Report, HSP-196, Dept. of Electrical Engineering, University of Edinburgh, May 1976.

[Rosenberg74] Rosenberg L. M. & Benbassat C.
CRITIC: An Integrated Circuit Design Rule Checking Program.
Proceedings of the 11th Design Automation Conference, 1974.

[Rowson78] Rowson J. A. & Wipfli J.
A MOS Logic Circuit Simulator.
Caltech SSP File #2200, 1978.

[Rowson79] Rowson J. A.
Understanding Hierarchical Design: A Thesis Proposal.
Caltech Memo #3238, 1979.

[Russell78] Russell G.
Automatic Mask Function Checking of LSI Circuits.
Proceedings of the 3rd International Conference on Computers in Engineering and Building Design, 1978.

[Smith80] Smith L. D.
The Elementary Structural Description Language.
CSR-53-80, Dept. of Computer Science, University of Edinburgh, 1980.

[Sproull80] Sproull R. F. & Lyon R. F.
The Caltech Intermediate Form for LSI Layout Description.
Contained in [Mead80].

[Sutherland76] Sutherland I. E., Mead C. A. & Everhart T. E.
Basic Limitations in Microcircuit Fabrication Technology.
R-1956-ARPA November 1976.

[Sutherland77] Sutherland I. E. & Mead C. A.
Microelectronics and Computer Science.
Scientific American, September 1977.

[Sutherland78] Sutherland I. E.
The Polygon Package.
Caltech Display File #1438, 1978.

[Stephens74] Stephens P. D.
The IMP Language and Compiler.

Computer Journal, Volume 17, No 3, 1974.

[Tarolli79] Tarolli G.
Towards a Working VLSI CAD Tool: A Chip Assembler.
Caltech SSP File #3131, 1979.

[Trimberger79] Trimberger S.
A CAD System Combining Interactive Graphics and a Layout Language.
Caltech SSP File #2499, 1979.

[vanCleemp77] vanCleemput W. M. & Slutz E. A.
Initial Design Considerations for a Hierarchical IC Design System.
Stanford University, Digital Systems Lab., Tech. Note No. 132, 1977.

[vanCleemp79] vanCleemput W. M.
Hierarchical Design for VLSI: Problems and Advantages.
Caltech Conference on VLSI, January 1979.

[Weinberger79] Weinberger A.
High Speed Programmable Logic Array Adders.
IBM Journal of Research and Development, Volume 23, Number 2, March 1979.

[Wilcox78] Wilcox P., Rombeek H. & Caughey D. H.
Design Rule Verification Based on One Dimensional Scans.
Proceedings of the 15th Design Automation Conference, 1978.

[Williams77] Williams J. D.
STICKS - A New Approach to LSI Design.
MIT M.Sc. Thesis, 1977.

[Wipfli78] Wipfli J.
A SIMULA Graphics Package.
Caltech SSP File #1929, 1978.

[Wood69] Wood J. et al
Computer Aided Production of Masks for Silicon Integrated Circuits.
IEE Conference Publication 51, 1969.

[Wulf73] Wulf W. & Shaw M.
Global Variables Considered Harmful.
SIGPLAN Notices, February 1973.

[Yoshida77] Yoshida K. et al
A Layout Checking System for Large Scale Integrated Circuits.
Proceedings of the 14th Design Automation Conference, 1977.

APPENDIX A

SIMULA Design Description Example

Shift Register Cell Array

```
B1          1  BEGIN
            2     EXTERNAL CLASS things,display,views,polsys,icsys;
            3     EXTERNAL INTEGER PROCEDURE xwd,lnot,lor,land;
            4     EXTERNAL TEXT PROCEDURE upcase,getitem,frontstrip,conc,initem,inline,scanto;
            5     EXTERNAL TEXT PROCEDURE front,rest;
            6     EXTERNAL CHARACTER PROCEDURE getch;
            7     EXTERNAL PROCEDURE enterdebug,sleep,split;
            8     EXTERNAL INTEGER PROCEDURE trmop,search;
            9
B2         10     icsys BEGIN
           11
           12
           13        REF (menu) defncom,simcom;
           14
           15        PROCEDURE defnoptions;
B3         16        BEGIN
           17           REF (blockdef) definst;
           18           TEXT deftext,cmd,yorn;
           19           REAL xx;
           20
B4         21           WHILE cmd\=defncom.opt(4) DO BEGIN
           22              cmd:-Copy(defncom.getcmd);
B5         23              IF cmd=defncom.opt(1) THEN BEGIN
           24                 Outtext ("name version:"); Breakoutimage;
           25                 deftext:-upcase (Copy (initem (inf)));
           26                 definst:-blockdefset.find (upcase (
           27                 Copy (inline (Copy ("name definition to instance:"),inf))));
B6         28                 IF definst==NONE THEN BEGIN
           29                    Outtext ("no definition of that name"); Outimage;
B7  E6     30                 END ELSE BEGIN
           31                    Outtext ("repetition in (x,y):"); Breakoutimage;
           32                    currentdef:-NEW srcellarraydef (Inint,Inint,
           33                    definst,deftext);
           34                    currentdef.setmaxwindow;
E7         35                 END;
B8  E5     36              END ELSE IF cmd=defncom.opt(2) THEN BEGIN
           37                 defncom.namedef (deftext,yorn);
B9         38                 IF yorn="Y" THEN BEGIN
           39                    Outtext ("gnd,sig,vdd,phi,xlim,ylim (2,6,21,13,21,26):");
           40                    Breakoutimage;
           41                    currentdef:-NEW srcelldef (Inint,Inint,Inint,Inint,
           42                    Inint,Inint,deftext);
E9         43                 END ELSE
           44                 currentdef:-NEW srcelldef (xx,xx,xx,xx,xx,xx,deftext);
           45                 currentdef.setmaxwindow;
E8         46              END ELSE IF cmd=defncom.opt(3) THEN defncom.helpinfo;
E4         47           END;
E3         48        END of defnoptions;
           49
           50        PROCEDURE initdefnopt;
B10        51        BEGIN
           52
```

```
            53          defncom:-NEW menu (4);
            54          defncom.opt(1):-Copy ("srcarray");
            55          defncom. help(1):-Copy
            56          ("srcarray - construct an array of shift register cells");
            57          defncom.opt(2):-Copy ("srcell");
            58          defncom.help(2):-Copy ("srcell - construct a shift register cell definition");
            59          defncom.opt(3):-Copy ("help");
            60          defncom.help(3):-Copy("help - print this information");
            61          defncom.opt(4):-Copy ("return");
            62          defncom.help(4):-Copy ("return - return to main command level");
            63
            64          defncom.prompt:-Copy ("definition");
            65
E10         66        END of initdefnopt;
            67
            68        PROCEDURE simoptions;
B11         69        BEGIN
            70          TEXT cmd;
            71          INTEGER i;
            72
B12         73          WHILE cmd\=simcom.opt(6) DO BEGIN
            74            cmd:-Copy(simcom.getcmd);
B13         75            IF cmd=simcom.opt(1) THEN BEGIN
B14         76              INSPECT currentdef DO BEGIN
            77                setstate ("gin").fixlo; setstate ("vin").fixhi;
            78                setstate ("phin").zero.one;
            79                setstate ("sigin").one.one.zero.zero;
E14         80              END;
            81              logsimblockdef (currentdef);
B15 E13     82            END ELSE IF cmd=simcom.opt(2) THEN BEGIN
B16         83              INSPECT currentdef DO BEGIN
B17         84                FOR i:=1 STEP 1 UNTIL currentdef QUA srcellarraydef.yrep DO BEGIN
            85                  setstate (sl("gin",i)).fixlo; setstate (sl("vin",i)).fixhi;
            86                  setstate (sl("sigin",i)).ones(4).zeros(4);
E17         87                END;
E18         88                FOR i:=1 STEP 1 UNTIL currentdef QUA srcellarraydef.xrep/2 DO BEGIN
            89                  setstate (sl("phi1in",i)).zero.one.zero.zero;
            90                  setstate (sl("phi2in",i)).zero.zero.zero.one;
E18         91                END;
E16         92              END;
            93              logsimblockdef (currentdef);
B19 E15     94            END ELSE IF cmd=simcom.opt(3) THEN BEGIN
            95              displayfull:=NOT displayfull; Outtext ("full node display ");
            96              IF displayfull THEN Outtext ("ON") ELSE Outtext ("OFF");
            97              Outimage;
B20 E19     98            END ELSE IF cmd=simcom.opt(4) THEN BEGIN
            99              currentdef.blocksimon:=NOT currentdef.blocksimon;
B21        100              IF currentdef.blocksimon THEN BEGIN
           101                displayfull:=FALSE; Outtext ("block on");
E21        102              END ELSE Outtext ("block off"); Outimage;
E20        103            END ELSE IF cmd=simcom.opt(5) THEN simcom.helpinfo;
E12        104          END;
```

```
E11     105         END of simoptions;
        106
        107         PROCEDURE initsimopt;
B22     108         BEGIN
        109            simcom:-NEW menu(6);
        110            simcom.opt(1):-Copy ("cell");
        111            simcom.help(1):-Copy ("cell - simulate srcell");
        112            simcom.opt(2):-Copy ("array");
        113            simcom.help(2):-Copy ("array - simulate srcell array");
        114            simcom.opt(3):-Copy ("full");
        115            simcom.help(3):-Copy ("full - full node display switch");
        116            simcom.opt(4):-Copy("block");
        117            simcom.help(4):-Copy("block - block level simulation on");
        118            simcom.opt(5):-Copy ("help");
        119            simcom.help(5):-Copy ("help - print this information");
        120            simcom.opt(6):-Copy ("return");
        121            simcom.help(6):-Copy ("return - back to command level");
        122
        123            simcom.prompt:-Copy ("simulate");
E22     124         END of initsimopt;
        125
        126         blockdef CLASS srcelldef (gnd, sig, vdd, phi, xlim, ylim,type);
        127         VALUE type; TEXT type;   REAL gnd,sig, vdd, phi, xlim, ylim;
B23     128         BEGIN
        129            INTEGER store0pass0,store0pass1,store1pass0,store1pass1,invbase,ypass;
        130
        131            PROCEDURE blocksim (nss,coord); REF (nodestateset) nss;
        132            REF (coordinode) coord;
B24     133            BEGIN
        134               IF nameof (coord,"vin") THEN simoutput (nss,"vout",valueof(nss,coord)) ELSE
        135               IF nameof (coord,"gin") THEN simoutput (nss,"gout",valueof (nss,coord))
        136               ELSE
B25     137               IF nameof (coord,"sigin") THEN BEGIN
B26     138                  IF valueof (nss,coord)>0 THEN BEGIN
        139                     IF nss.blockstate=store0pass0 THEN nss.blockstate:=store1pass1 ELSE
B27     140                     IF nss.blockstate=store0pass0 THEN BEGIN
        141                        nss.blockstate:=store1pass1; simoutput (nss,"sigout",zeroed);
E27     142                     END ELSE
        143                     IF nss.blockstate=store1pass1 THEN simoutput (nss,"sigout",zeroed);
B28 E26 144                  END ELSE BEGIN
        145                     IF nss.blockstate=store0pass1 THEN simoutput (nss,"sigout",oned) ELSE
        146                     IF nss.blockstate=store1pass0 THEN nss.blockstate:=store0pass0 ELSE
B29     147                     IF nss.blockstate=store1pass1 THEN BEGIN
        148                        nss.blockstate:=store0pass1; simoutput (nss,"sigout",oned);
E29     149                     END;
E28     150                  END;
E30 E25 151               END ELSE IF nameof (coord,"phin") THEN BEGIN
B31     152                  IF valueof (nss,coord)>0 THEN BEGIN
        153                     IF nss.blockstate=store0pass0 THEN simoutput (nss,"sigout",oned) ELSE
        154                     IF nss.blockstate=store1pass1 THEN simoutput (nss,"sigout",zeroed) ELSE
B32     155                     IF nss.blockstate=store0pass0 THEN BEGIN
        156                        nss.blockstate:=store0pass1; simoutput (nss,"sigout",oned);
```

```
B33 E32   157                    END ELSE IF nss.blockstate=storelpass0 THEN BEGIN
          158                       nss.blockstate:=storelpass1; simoutput (nss,"sigout",zeroed);
E33       159                    END;
B34 E31   160                 END ELSE BEGIN
          161                    IF nss.blockstate=store0pass1 THEN nss.blockstate:=store0pass0 ELSE
          162                    IF nss.blockstate=storelpass1 THEN nss.blockstate:=storelpass0;
E34       163                 END;
          164                 simoutput (nss,"phout",valueof (nss,coord));
E30       165              END;
E24       166           END of blocksim;
          167
          168           !set text name of cell definition;
          169           nameblock (conc ("srcell.",type));
          170
          171           ! check default settings for parameters;
          172           default (gnd,2); default (sig,6); default (vdd,21);
          173           default (phi,13); default (xlim,21); default (ylim,26);
          174           !define block bounding box;
          175           blockboundbox (0,0,xlim, ylim);
          176
          177           !define the inverter x baseline and;
          178           !the pass transistor y baseline;
          179           invbase:= 5; ypass:=sig+6;
          180
          181           !external interface;
          182           pin("gin");pin("gout");pin("vin");pin("vout");
          183           pin("phin");pin("phout");pin("sigin");pin("sigout");
          184
          185           !main components;
          186           passtran('ps");pulldown("inv");pullup("pull","buttx");
          187
          188           !wires;
          189           !power and ground lines;
          190           wire("gin","gx",metal).width (4); wire("gx","gout",metal).width (4);
          191           wire("vin","vx",metal).width (4); wire("vx","vout",metal).width (4);
          192
          193           !clock line;
          194           wire("phin","ps.in",poly);
          195           wire("ps.out","phout",poly);
          196
          197           !pullup connections;
          198           wire("vx","pull.drn",diff);
          199           wire("pull.src","buttx",diff);
          200
          201           !pulldown connections;
          202           wire("buttx","inv.drn",diff);
          203           wire("inv.src","gx",diff);
          204
          205           !data line wires;
          206           wire("sigin","inv.in",poly);
          207           wire("buttx","ps.src",diff).path.dy(-2).dx(5).ythenx;
          208           wire("ps.drn","psx",diff).path.xtheny;
```

```
209             wire("psx","sigout",poly).path.ythenx;
210
211             !coordinodes with contact attributes;
212             contact("gx");contact("vx");contact ("buttx").orientation (0,1);
213             contact("psx").orientation (0,-1);
214
215             !coordinode physical positions;
216
217             !power and ground lines;
218             place("gin",NEW point (0,gnd));
219             place("gx",NEW point (invbase,gnd));
220             place("gout",NEW point(xlim,gnd));
221             place("vin",NEW point(0,vdd));
222             place("vx",NEW point(invbase,vdd));
223             place("vout",NEW point(xlim,vdd));
224
225             !data and control pins;
226             place("sigin",NEW point (0,sig));
227             place("sigout",NEW point (xlim,sig));
228             place("phin",NEW point (phi,ylim));
229             place("phout",NEW point (phi,0));
230
231             !pullup connections;
232             place("pull.drn",NEW point (invbase,vdd-3));
233             place("pull.src",NEW point (invbase,vdd-10));
234
235             !inverter output to pass transistor;
236             place("buttx",NEW point (invbase,vdd-10));
237
238             !pulldown connections;
239             place("inv.drn",NEW point (invbase,sig+1));
240             place("inv.src",NEW point (invbase,sig-1));
241             place("inv.in",NEW point (invbase-3,sig));
242             place("inv.out",NEW point (invbase+3,sig));
243
244             !pass transistor connection;
245             place("ps.in",NEW point (phi,ypass+1));
246             place("ps.out",NEW point (phi,ypass-1));
247             place("ps.drn",NEW point (phi+1,ypass));
248             place("ps.src",NEW point (phi-1,ypass));
249
250             !signal connection output;
251             place("psx",NEW point (xlim-3,sig+4));
252
253             ! set up internal state code values (arbitrary);
254             store0pass0:=1; store0pass1:=2; store1pass0:=3; store1pass1:=4;
255
256
E23         257         END of srcelldef;
258
259
260
```

```
        261
        262         blockdef CLASS srcellarraydef (xrep,yrep,celldef,type);
        263         VALUE xrep,yrep,type; TEXT type; REAL xrep,yrep; REF(srcelldef) celldef;
B35     264         BEGIN
        265            INTEGER i,j,xcell,ycell,xrep2;
        266
        267            !set text name of cell definition;
        268            nameblock(conc ("srcellarray.",type));
        269
        270            !same useful numbers;
        271            xrep2:=xrep//2;
        272            xcell:=celldef.xlim;ycell:=celldef.ylim;
        273
        274            !set block bounding box to enable;
        275            !coordinode position checks;
        276            blockboundbox (0,0,xcell*xrep,ycell*yrep);
        277
        278            !pin specification;
        279
B36     280            FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        281               pin(s1("phi1in",i));pin(s1("phi1out",i));
        282               pin(s1("phi2in",i));pin(s1("phi2out",i));
E36     283            END;
        284
B37     285            FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        286               pin(s1("gin",i));pin(s1("gout",i));
        287               pin(s1("sigin",i));pin(s1("sigout",i));
        288               pin(s1("vin",i));pin(s1("vout",i));
E37     289            END;
        290
        291
        292            !shift register cell instance specification;
        293
B38     294            FOR i:=1 STEP 1 UNTIL xrep DO BEGIN
        295               FOR j:=1 STEP 1 UNTIL yrep DO
        296               blockinst(s2("src",i,j),celldef,
        297               NEW transform (NONE).translatedby (NEW point
        298               ((i-1)*xcell,(j-1)*ycell)));
E38     299            END;
        300
        301            !internal wire specification;
        302
        303            !power, ground and data lines;
        304
B39     305            FOR j:=1 STEP 1 UNTIL yrep DO BEGIN
B40     306               FOR i:=1 STEP 1 UNTIL xrep-1 DO BEGIN
        307                  wire(conc(s2("src",i,j),".gout"),conc(s2("src",i+1,j),".gin"),metal);
        308                  wire(conc(s2("src",i,j),".vout"),conc(s2("src",i+1,j),".vin"),metal);
        309                  wire(conc(s2("src",i,j),".sigout"),conc(s2("src",i+1,j),".sigin"),poly);
E40     310               END;
        311
E39     312            END;
```

```
        313
        314             ! clock lines;
        315
B41     316             FOR i:=1 STEP 1 UNTIL xrep DO BEGIN
        317                FOR j:=1 STEP 1 UNTIL yrep-1 DO
        318                wire(conc(s2("src",i,j+1),".phout"),conc(s2('src",i,j),".phin"),poly);
        319
E41     320             END;
        321
        322             !wires to pin;
B42     323             FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        324                wire(sl("gin",i),conc(s2("src",1,i),".gin"),metal);
        325                wire(sl("sigin",i),conc(s2("src",1,i),".sigin"),poly);
        326                wire(sl("vin",i),conc(s2("src",1,i),".vin"),metal);
        327                wire(sl("gout",i),conc(s2("src",xrep,i),".gout"),metal);
        328                wire(sl("sigout",i),conc(s2("src",xrep,i),".sigout"),poly);
        329                wire(sl("vout",i),conc(s2("src",xrep,i),".vout"),metal);
        330
E42     331             END;
        332
B43     333             FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        334                wire(sl("phi1in",i),conc(s2("src",2*i-1,yrep),".phin"),poly);
        335                wire(sl("phi1out",i),conc(s2("src",2*i-1,1),".phout"),poly);
        336                wire(sl("phi2in",i),conc(s2("src",2*i,yrep),".phin"),poly);
        337                wire(sl("phi2out",i),conc(s2("src",2*i,1),".phout"),poly);
E43     338             END;
        339
        340             !place pins physically;
        341
B44     342             FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        343                place(sl("gin",i),position(conc(s2("src",1,i),".gin")));
        344                place(sl("sigin",i),position(conc(s2("src",1,i),".sigin")));
        345                place(sl("vin",i),position(conc(s2("src",1,i),".vin")));
        346                place(sl("gout",i),position(conc(s2("src",xrep,i),".gout")));
        347                place(sl("sigout",i),position(conc(s2("src",xrep,i),".sigout")));
        348                place(sl("vout",i),position(conc(s2("src",xrep,i),".vout")));
E44     349             END;
        350
B45     351             FOR i:=1 STEP 1 UNTIL xrep2 DO BEGIN
        352                place(sl("phi1in",1),position(conc(s2("src",2*i-1,yrep),".phin")));
        353                place(sl("phi1out",i),position(conc(s2("src",2*i-1,1),".phout")));
        354                place(sl("phi2in",i),position(conc(s2("src",2*i,yrep),".phin")));
        355                place(sl("phi2out",i),position(conc(s2("src",2*i,1),".phout")));
E45     356             END;
        357
E35     358          END of srcellarraydef;
        359
        360          initdefnopt; initsimopt; mainoptions (defnoptions,simoptions);
        361
E2      362       END;
E1      363    END of program;
```

APPENDIX B

SIMULA Design Description Example

OM2 ALU Data Path

```
B1          1   BEGIN
            2      EXTERNAL CLASS things,displa,vews,icplot;
            3      EXTERNAL INTEGER PROCEDURE xwd,lnot,lor,land;
            4      EXTERNAL TEXT PROCEDURE upcase,getitem,frontstrip,conc,initem,inline,scanto;
            5      EXTERNAL TEXT PROCEDURE front,rest;
            6      ! see page 3-241 DECSYSTEM-20 Monitor Calls Manual ;
            7      EXTERNAL CHARACTER PROCEDURE getch;
            8      EXTERNAL PROCEDURE enterdebug,sleep,split;
            9      EXTERNAL INTEGER PROCEDURE trmop,search;
           10
B2         11      icplot BEGIN
           12
           13
           14         !*************************************************************;
           15         !          menu and control of block building          ;
           16         !*************************************************************;
           17
           18         REF (menu) defncom;
           19         REAL vdd,gnd,bus1,bus2,ylim;
           20
           21         PROCEDURE defnoptions;
B3         22         BEGIN
           23            TEXT deftext,yorn;
           24            REAL xx;
           25            BOOLEAN twomem,memtype,amp;
           26            REF (memcell) mem1,mem2;
           27            REF (aluinputdef) input;
           28            REF (alucarrychaindef) carry;
           29            REF (aluoutputdef) output;
           30            REF (om2bitslice) definst;
           31            SWITCH com:=me,inp,ca,ou,sl,ar,he,ba,re;
           32
           33            next:GOTO com(defncom.getcmd);
           34
           35            ba:Outtext ("change baselines -"); Outimage;
           36            defncom.getparm ("vdd",vdd); defncom.getparm ("gnd",gnd);
           37            defncom.getparm ("bus1",bus1); defncom.getparm ("bus2",bus2);
           38            defncom.getparm ("ylim",ylim); GOTO next;
           39
           40            me:defncom.namedef (deftext,yorn);
B4         41            IF yorn="Y" THEN BEGIN
           42               Outtext ("not in yet"); Outimage;
E4         43            END;
           44            Outtext ("stage 1 or 2:"); Breakoutimage; memtype:=Inint=2;
           45            currentdef:-NEW memcell (xx,xx,xx,xx,xx,xx,xx,memtype,deftext);
           46            currentdef.setmaxwindow; GOTO next;
           47
           48            inp:defncom.namedef (deftext,yorn);
B5         49            IF yorn="Y" THEN BEGIN
           50               Outtext ("not in yet"); Outimage;
E5         51            END;
           52            currentdef:-NEW aluinputdef (xx,xx,xx,deftext);
```

```
                53          currentdef.setmaxwindow; GOTO next;
                54
                55          ca:defncom.namedef (deftext,yorn);
B6              56          IF yorn="Y" THEN BEGIN
                57             Outtext ("not in yet"); Outimage;
E6              58          END;
                59          Outtext ("amplifier stage (0=no,1=yes):"); Breakoutimage;
                60          amp:=Inint=1;
                61          currentdef:-NEW alucarrychaindef (xx,xx,xx,amp,deftext);
                62          currentdef.setmaxwindow; GOTO next;
                63
                64          ou:defncom.namedef (deftext,yorn);
B7              65          IF yorn="Y" THEN BEGIN
                66             Outtext ("not in yet"); Outimage;
E7              67          END;
                68          currentdef:-NEW aluoutputdef (xx,xx,xx,xx,xx,xx,deftext);
                69          currentdef.setmaxwindow; GOTO next;
                70
                71          s1:Outtext ("name version:"); Breakoutimage;
                72          deftext:-upcase (Copy (initem (inf)));
                73          Outtext ("memory or alu (1 or 2):"); Breakoutimage; twomem:=Inint=1;
B8              74          IF twomem THEN BEGIN
                75             mem1:-findcelldef ("memory cell 1");
                76             mem2:-findcelldef ("memory cell 2");
B9  E8          77          END ELSE BEGIN
                78             input:-firdcelldef ("input cell");
                79             carry:-firdcelldef ("carry cell");
                80             output:-findcelldef ("output cell");
E9              81          END;
                82          IF (twomem AND (mem1==NONE OR mem2==NONE))
                83          OR (NOT twomem AND (input==NONE OR carry==NONE OR output==NONE)) THEN
B10             84          BEGIN
                85             Outtext ("parameter unspecified"); Outimage; GOTO next;
E10             86          END;
                87          currentdef:-NEW om2bitslice
                88          (mem1,mem2,input,carry,output,twomem,deftext);
                89          currentdef.setmaxwindow; GOTO next;
                90
                91          ar:Outtext ("name version:"); Breakoutimage;
                92          deftext:-upcase (Copy (initem (inf)));
                93          definst:-findcelldef ("aluslice cell");
B11             94          IF definst==NONE THEN BEGIN
                95             Outtext ("no definition of that name"); Outimage; GOTO next;
E11             96          END;
                97          Outtext ("alu width:"); Breakoutimage;
                98          currentdef:-NEW om2aludef (Inint,definst,twomem,deftext);
                99          GOTO next;
               100
               101          he:defncom.helpinfo;
               102
               103          re:
               104
```

```
E3      105         END of defnoptions;
        106
        107         PROCEDURE initdefnopt;
B12     108         BEGIN
        109            defncom:-NEW menu (9);
        110            defncom.opt (1):-Copy("memory");
        111            defncom.opt (2):-Copy("input");
        112            defncom.opt (3):-Copy("carry");
        113            defncom.opt (4):-Copy("output");
        114            defncom.opt(5):-Copy ("slice");
        115            defncom.opt (6):-Copy ("path");
        116            defncom.opt (7):-Copy("help");
        117            defncom.opt (8):-Copy ("baselines");
        118            defncom.opt (9):-Copy("return");
        119            defncom.help(1):-Copy ("memory - construct dual port register block");
        120            defncom.help(2):-Copy ("input - construct alu input stage");
        121            defncom.help(3):-Copy ("carry - construct alu carry stage");
        122            defncom.help(4):-Copy ("output - construct alu output stage');
        123            defncom.help (5):-Copy ("slice - construct an alu slice");
        124            defncom.help(6):-Copy ("path - construct an N wide alu data path");
        125            defncom.help(7):-Copy ("help - print this information");
        126            defncom.help(8):-Copy ("baselines - define placement base lines");
        127            defncom.help(9):-Copy ("return - return to main command level");
        128
        129            defncom.prompt:-Copy ("definition");
        130
E12     131         END of initdefnopt;
        132
        133         PROCEDURE simoptions;
B13     134         BEGIN
E13     135         END of simoptions;
        136
        137         PROCEDURE initsimopt;
B14     138         BEGIN
E14     139         END of initsimopt;
        140
        141         !*****************************************************************;
        142         !              alu memory dual port register cell                 ;
        143         !*****************************************************************;
        144
        145         blockdef CLASS memcell
        146         (read1,read2,drive1,drive2,refresh,vddinv,xlim,mem2,type);
        147         VALUE type,mem2; TEXT type;
        148         BOOLEAN mem2;
        149         REAL read1,read2,drive1,drive2,refresh,vddinv,xlim;
        150
B15     151         BEGIN
        152
        153            REAL invdat,refdat;
        154
        155            nameblock(conc ("memory.",type));
        156
```

```
                default (read1,11);
                default (read2,3);
                default (drive1,35);
                default (drive2,42);
                default (refresh,15);
                default (vddinv,19);
                default (xlim,43);

                blockboundbox(0,0,xlim,ylim);

                invdat:=bus2+9.5;refdat:=bus1-9.5;


                !pin specification;
                pin("vddin");pin("vddout");pin("gndin");
                pin("gndout");pin("inbus1");pin("outbus1");
                pin("inbus2");pin("outbus2");
                IF mem2 THEN pin("indrbus2") ELSE pin("outdrbus2");
                pin("rdinbus2");pin("rdoutbus2");
                pin("rdinbus1");pin("rdoutbus1");
                pin("drinbus1");pin("droutbus1");
                pin("drinbus2");pin("droutbus2");
                pin("refin");pin("refout");
                pin("vddinvin");pin("vddinvout");

                ! memory outputs to alu input stage;
                pin ("zoverout"); pin ("zbaroverout");
B16             IF mem2 THEN BEGIN
                   pin ("zout"); pin ("zoverin"); pin ("zbarout"); pin ("zbaroverin");
E16             END;

        !NB a single contact doubles for both read drive data access
        to bus 2 between successive cells of an array;

                !main components;

                !bus read and drive pass transistors;
                passtran("psrd1");passtran("psrd2");
                passtran("psdr1").intoout;
                passtran("psdr2").intoout.xy(drive2-3,bus2-2).
                dy(4);

                !refresh pass transistor;
                passtran("ref");

                !inverter pullups and pull downs;
                pullup("pull1","meminx").srctodrn.dy(-7).<theny;
                pulldown("inv1").intoout.width(2.818);
                pullup("pull2","memoutx").srctodrn.ythenx;
                pulldown("inv2").intoout.dy(2).dxy(-3,3).dx(-3);

                !power and ground wiring;
```

```
209             wire("vddin","vx",metal).width (4);
210             wire("vx","vddout",metal).width (4);
211             wire("gndin","gx",metal).width (4).path.xtheny;
212             wire ("gndout","gx",metal).width (4).path.xtheny;
213             wire("vx","vddinvin",diff).path.xtheny;
214
215             !read and drive control wires and refresh wires;
216             wire("rdinbus2","psrd2.in",poly).path.y(bus2-4.5).xtheny;
217             wire("psrd2.out","rdoutbus2",poly).path.dy(3).xtheny;
218             wire("rdinbus1","psrd1.in",poly).path.y(bus1-4.5).xtheny;
219             wire("psrd1.out","rdoutbus1",poly).path.dy(3).xtheny;
220             wire("drinbus1","fix",poly).path.y(bus2-6.5).dxy(-1,1).
221             y(bus2+5.5).dxy(1,1).y(gnd);
222             wire
223             ("psdr1.in","fix",poly).path.dy(-2).dx(-4).dy(-5).dxy(4,-4).y(gnd);
224             wire("psdr1.out","droutbus1",poly);
225             wire("psdr2.in","drinbus2",poly).path.dxy (2,-2).y(0);
226             wire("psdr2.out","droutbus2",poly).path.dxy(2,2).y(ylim);
227             wire("refin","ref.in",poly).path.y(refdat-4).xtheny;
228             wire("ref.out","refout",poly).path.dy(5).dx(-2).y(bus1+3.5).xtheny;
229
230
231             !bus lines;
232             wire("rdbus1x","inbus1",metal).path.ythenx;
233             wire("rdbus1x","drbus1x",metal).path.y(bus1).xtheny;
234             wire("drbus1x","outbus1",metal).path.ythenx;
235             wire("rdrbus2x","inbus2",metal).path.ythenx;
236             wire("rdrbus2x","outbus2",metal).path.ythenx;
237
238             !fiddled use of bus 2 contact;
239             IF mem2 THEN wire("rdrbus2x","indrbus2",diff).path.ythenx
240             ELSE wire("outdrbus2","psdr2.drn",diff);
241
242
243             !connect read lines to first inverter and buses;
244             wire("rdrbus2x","psrd2.drn",diff);
245             wire("psrd2.src","toinv1x",diff).path.x(read2+5).ythenx;
246             wire("toinv1x","toreflx",diff);
247             wire("toreflx","psrd1.src",diff).path.dx(-1).ythenx;
248             wire("toinv1x","togtlx",metal).path.dy(0.5).xtheny;
249             wire("inv1.in","togtlx",poly).path.dxy(-1,-1).ythenx;
250             wire("psrd1.drn","rdbus1x",diff).path.xtheny;
251
252             !connect drive lines to buses;
253             wire("memoutx","frominv2x",metal);
254             wire("frominv2x","psdr1.src",diff).path.dx(1).ythenx;
255             wire("psdr1.drn","drbus1x",diff).path.xtheny;
256             wire("frominv2x","psdr2.src",diff).path.xtheny;
257
258             !NB bus 2 drive line already connected to pin;
259
260             !connect refresh line to input;
```

```
          261           wire("memoutx","ref.src",diff).path.ythenx;
          262           wire("ref.drn","toref2x",diff).path.xtheny;
          263           wire("toref2x","toref1x",metal).path.dy(-0.5).xtheny;
          264
          265           !connect up first inverter;
          266           wire("vx","pull1.drn",diff).path.xtheny;
          267           wire("pull1.src","meminx",diff);
          268           wire("inv1.drn","meminx",diff).path.xy(vddinv+6,invdat+1).xtheny;
          269           wire("meminx","inv2.in",poly).path.dy(-1).xtheny;
          270           wire("inv1.src","gx",diff).path.dxy(-3,3).ythenx;
          271           !connect up second inverter;
          272           wire("pull2.drn","vddinvout",diff).path.xtheny;
          273           wire("pull2.src","memoutx",diff);
          274           wire("inv2.drn","memoutx",diff).path.ythenx;
          275           wire("inv2.src","gx",diff).path.y(gnd).xtheny;
          276
          277           ! connect ouputs to alu input stage;
B17       278           IF mem2 THEN BEGIN
          279              wire ("frominv2x","zout",metal).path.ythenx;
          280              wire ("zoverin","zoverout",metal);
          281              wire ("meminx","zbarout",metal).path.ythenx;
          282              wire ("zbaroverin","zbaroverout",metal);
B18 E17   283           END ELSE BEGIN
          284              wire ("frominv2x","zoverout",metal).path.ythenx;
          285              wire ("meminx","zbaroverout",metal).path.ythenx;
E18       286           END;
          287
          288           !make contacts explicit;
          289
          290           !power and ground contacts;
          291           contact("gx");contact("vx");
          292
          293           !contacts to buses for read and drive lines;
          294           contact("rdbus1x");contact("drbus1x");
          295           contact("rdrbus2x");
          296
          297           !contacts from bus reads to inverter and refresh;
          298           ! and from inverter to drive line;
          299           contact ("toref1x"); contact ("toinv1x"); contact ("togt1x');
          300           contact ("toref2x"); contact("frominv2x');
          301
          302           !butting contacts at pullups;
          303           contact("meminx").orientation(0,-1);
          304           contact("memoutx").orientation(0,1);
          305
          306       !define coordinode positions with respect to
          307        parameterised baselines;
          308
          309           !power and ground coordinode postions;
          310           place("vddin",NEW point(0,vdd));
          311           place("vx",NEW point(vddinv,vdd));
          312           place("vddout",NEW point(xlim,vdd));
```

```
313             place("gndin",NEW point(0,gnd));
314             place("gx",NEW point(vddinv,gnd-1));
315             place("gndout",NEW point(xlim,gnd));
316
317             !bus line coordinode positions;
318             place("inbus1",NEW point(0,bus1));
319             place("drbus1x",NEW point(drive1-4,bus1+0.5));
320             place("rdbus1x",NEW point(read1+2,bus1-0.5));
321             place("outbus1",NEW point(xlim,bus1));
322             place("inbus2",NEW point(0,bus2));
323             place("rdrbus2x",NEW point(read2-2,bus2+0.5));
324             place("outbus2",NEW point(xlim,bus2));
325
326             !fiddled bus 2 data line positions;
327             IF mem2 THEN place("indrbus2",NEW point(0,bus2))
328             ELSE place("outdrbus2",NEW point(xlim,bus2));
329
330             !refresh line coordinodes;
331             place("refin",NEW point(refresh,0));
332             place("ref.in",NEW point(refresh+4,refdat-2));
333             place ("ref.out",NEW point (refresh+4,refdat));
334             place("refout",NEW point(refresh,ylim));
335
336             ! coordinodes for power line to pullups;
337             place ("vddinvin", NEW point (vddinv+1,0));
338             place ("vddinvout",NEW point (vddinv+1,ylim));
339
340         !coordinodes for read line joining to
341          first inverter buses and refresh input;
342             place("psrd1.drn", NEW point(read1-1,bus1-0.5));
343             place("psrd1.src",NEW point(read1-3,bus1-0.5));
344             place("toref1x",NEW point(read1-4,refdat+1));
345             place("toref2x",NEW point(read1+4,refdat+1));
346             place("ref.src", NEW point(refresh+5,refdat-1));
347             place("ref.drn",NEW point(refresh+3,refdat-1));
348
349             !read input to first inverter;
350             place("toinv1x", NEW point(read1-4,invdat-1));
351             place("togt1x",NEW point(vddinv+1,invdat-2));
352             place("psrd2.drn",NEW point(read2+1,bus2+0.5));
353             place("psrd2.src",NEW point(read2+3,bus2+0.5));
354
355             !read line coordinodes;
356             place("rdinbus2",NEW point(read2,0));
357             place("psrd2.in",NEW point(read2+2,bus2-0.5));
358             place("psrd2.out",NEW point(read2+2,bus2+1.5));
359             place("rdoutbus2",NEW point(read2,ylim));
360             place("rdinbus1",NEW point(read1,0));
361             place("psrd1.in",NEW point(read1-2,bus1-1.5));
362             place("psrd1.out",NEW point(read1-2,bus1+0.5));
363             place("rdoutbus1",NEW point(read1,ylim));
364
```

```
365             !drive line coordinodes;
366             place("drinbus1",NEW point(drive1,0));
367             place ("fix",NEW point (drive1,gnd));
368             place("psdr1.in",NEW point(drive1,bus1-1.5));
369             place("psdr1.out",NEW point(drive1,bus1+4.5));
370             place("droutbus1",NEW point(drive1,ylim));
371             place("drinbus2",NEW point(drive2,0));
372             place("psdr2.in",NEW point(drive2-2,bus2-3));
373             place("psdr2.out",NEW point(drive2-2,bus2+3));
374             place("droutbus2",NEW point(drive2,ylim));
375
376             !coordinodes for write lines;
377             place("frominv2x",NEW point(drive1+2,bus1-7.5));
378             place("psdr1.src",NEW point(drive1+1,bus1-0.5));
379             place("psdr1.drn",NEW point(drive1-1,bus1-0.5));
380             place("psdr2.src",NEW point(drive2-4,bus2));
381             place("psdr2.drn",NEW point(drive2-2,bus2));
382
383             !coordinodes for double inverter circuit;
384             place("pull1.drn",NEW point(vddinv+2,bus2-3.5));
385             place("pull1.src",NEW point(vddinv+9,bus2+7.5));
386             place("meminx",NEW point(vddinv+8,bus2+7.5));
387             place("inv1.in",NEW point(vddinv+3,invdat+2));
388             place("inv1.out",NEW point(vddinv+5,invdat+4));
389             place("inv1.drn",NEW point(vddinv+5,invdat+2));
390             place("inv1.src",NEW point(vddinv+3,invdat+4));
391             place("inv2.src",NEW point(vddinv+6,refdat-5));
392             place("inv2.drn",NEW point(vddinv+6,refdat-3));
393             place("inv2.in",NEW point(vddinv+11,refdat-9));
394             place("inv2.out",NEW point(vddinv+5,refdat-4));
395             place("memoutx",NEW point(vddinv+5,bus1-7.5));
396             place("pull2.src",NEW point(vddinv+6,bus1-7.5));
397             place("pull2.drn",NEW point(vddinv+3,bus1+4.5));
398
399             ! place connections to alu input stage;
400             place ("zoverout",NEW point (xlim,bus1-15));
401             place ("zbaroverout",NEW point (xlim,bus2+15));
B19     402     IF mem2 THEN BEGIN
403                place ("zout",NEW point (xlim,bus1-7));
404                place ("zoverin",NEW point (0,bus1-15));
405                place ("zbarout",NEW point (xlim,bus2+8));
406                place ("zbaroverin",NEW point (0,bus2+15));
E19     407     END;
408
E15     409     END of memcell;
410
411     !*********************************************************;
412     !              alu input stage                            ;
413     !*********************************************************;
414
415     blockdef CLASS aluinputdef (xlim,pkst,pkinc,type);
416     VALUE type; TEXT type; REAL xlim,pkst,pkinc;
```

```
B20     417         BEGIN
        418            REAL pff,ptt,ktt,ktf,ptf,pft,kft,kff,pxbar,tfbar,ttbar,ftbar;
        419
        420            nameblock (conc ("input.",type));
        421
        422            ! set up defaults if necessary;
        423            default (xlim,82);   default (pkst,23);   default (pkinc,7);
        424
        425            ! set up other derived baselines;
        426            pff:=pkst+2; ptt:=pkst+pkinc+1; ktt:=pkst+2*pkinc; ktf:=pkst+3*pkinc-3;
        427            ptf:=pkst+4*pkinc+1; pft:=pkst+5*pkinc-2; kft:=pkst+6*pkinc-3;
        428            kff:=pkst+7*pkinc-2; pxbar:=bus2+7; tfbar:=bus2+20; ttbar:=bus1-15;
        429            ftbar:=bus1-7;
        430
        431            ! set up bounding box of block's physical description;
        432            blockboundbox (0,0,xlim,ylim);
        433
        434            !*****************************************************************;
        435            !               external pin specifications                      ;
        436            !*****************************************************************;
        437
        438            ! carry and propogate control;
        439            pin ("pffin"); pin ("pffout"); pin("pttin"); pin ("pttout");
        440            pin ("kttin"); pin ("kttout"); pin ("ktfin"); pin ("ktfout");
        441            pin ("ptfin"); pin ("ptfout"); pin ("pftin"); pin ("pftout");
        442            pin ("kftin"); pin ("kftout"); pin ("kffin"); pin ("kffout");
        443
        444            ! data and complement inputs;
        445            pin ("bin"); pin ("ain"); pin ("abarin"); pin ("bbarin");
        446
        447            ! power, ground and busses;
        448            pin ("gndin"); pin ("gndout"); pin ("vddin"); pin ("vddout");
        449            pin ("inbus1"); pin ('outbus1"); pin ("inbus2"); pin ("outbus2");
        450
        451            ! carry and propogate output;
        452            pin ("kbar"); pin ("pbar");
        453
        454            ! connect vdd on diff;
        455            pin ("vddiffin"); pin ("vddiffout");
        456
        457            !*****************************************************************;
        458            !          transistor components                                 ;
        459            !*****************************************************************;
        460
        461            ! input permutations;
        462            pulldown ("abarftgt"); pulldown ("bftgt"); pulldown ("bttgt");
        463            pulldown ("attgt"); pulldown ("atfgt"); pulldown ("bbartfgt");
        464            pulldown ("abarffgt"); pulldown ("bbarffgt");
        465
        466            ! kill and propogate controls;
        467            pulldown ("pffgt"); pulldown ("pttgt"); pulldown ("kttgt");
        468            pulldown ("ktfgt"); pulldown ("ptfgt"); pulldown ("pftgt");
```

```
469         pulldown ("kftgt"); pulldown ("kffgt");
470
471         ! pullups;
472         pullup ("killp","killx"); pullup ("propp","propx");
473
474         !*************************************************************;
475         !          wire up the components                             ;
476         !*************************************************************;
477
478         ! power and ground wiring;
479         wire ("vddin","vx",metal).width (4);
480         wire ("vx","vddout",metal).width (4);
481         wire ("vx","vddiffout",diff);
482         wire ("gndin","gx",metal).path.xtheny.width (4);
483         wire ("gx","gndout",metal).width (4).path.ythenx;
484
485         ! bus wiring;
486         wire ("inbus1","outbus1",metal);
487         wire ("inbus2","outbus2",metal);
488
489         ! data input permutation generators;
490         wire ("bin","bx",metal);
491         wire ("bx","bftgt.in",poly).path.xtheny;
492         wire ("bx","bttgt.in",poly).path.xtheny;
493         wire ("ain","ax",metal);
494         wire ("ax","attgt.in",poly).path.xtheny;
495         wire ("attgt.out","atfgt.in",poly);
496         wire ("abarin","abarx",metal);
497         wire ("abarx","abarftgt.in",poly).path.dx(-1).ythenx;
498         wire ("abarx","abarffgt.in",poly).path.ythenx;
499         wire ("bbarin","bbarx",metal);
500         wire ("bbarx","bbarffgt.in",poly).path.ythenx;
501         wire ("bbarffgt.out","bbartfgt.in",poly).path.xtheny;
502
503         ! wires to A B nand outputs;
504         wire ("ftx1","bftgt.drn",diff).path.ythenx;
505         wire ("bftgt.src","abarftgt.drn",diff).path.xtheny;
506         wire ("abarftgt.src","gx",diff).path.ythenx;
507         wire ("ttx1","attgt.drn",diff).path.ythenx;
508         wire ("attgt.src","bttgt.drn",diff);
509         wire ("bttgt.src","gx",diff).path.dx(-2).ythenx;
510         wire ("tfx1","bbartfgt.drn",diff).path.ythenx;
511         wire ("bbartfgt.src","atfgt.drn",diff);
512         wire ("atfgt.src","gx",diff);
513         wire ("bbarffgt.drn","ffx1",diff).path.dy(-2).xtheny;
514         wire ("bbarffgt.src","abarffgt.drn",diff);
515         wire ("abarffgt.src","gx",diff).path.ythenx;
516
517         ! wires to kill and propogate pass transistors;
518         wire ("ftx1","ftx2",metal);
519         wire ("ftx2","ftfork",diff).path.xtheny;
520         wire ("ftfork","pftgt.src",diff).path.ythenx;
```

```
521           wire ("kftgt.drn","kx1",diff);
522           wire ("pftgt.drn","px1",diff);
523           wire ("ttx1","ttx2",metal);
524           wire ("ttx2","pttgt.src",diff);
525           wire ("pttgt.drn","px2",diff).path.dx(-2).dy(-7).xtheny;
526           wire ("ttx2","kttgt.src",diff);
527           wire ("kttgt.drn","kx2",diff);
528           wire ("tfx1","tfx2",metal);
529           wire ("tfx2","ktfgt.src",diff);
530           wire ("ktfgt.drn","kx2",diff);
531           wire ("tfx2","ptfgt.drn",diff).path.ythenx;
532           wire ("ptfgt.src","px1",diff);
533           wire ("ffx1","pffgt.src",diff);
534           wire ("pffgt.drn","px2",diff);
535           wire ("ffx1","ffx2",metal).path.dy(7).x(pff+1).ythenx;;
536           wire ("ffx2","kffgt.src",diff);
537           wire ("kffgt.drn","kx1",diff).path.xtheny;
538
539           ! kill propogate collect points to pullups;
540           wire ("kx1","killx",metal).path.xtheny;
541           wire ("kx2","killx",metal).path.xtheny;
542           wire ("px1","propx",metal).path.xtheny;
543           wire ("px2","propx",metal).path.xtheny;
544
545           ! wires from pullup outputs to pins;
546           wire ("killx","kbar",pcly).path.dy(1).dx(4).ythenx;
547           wire ("propx","pbar",pcly).path.dy(-1).dx(4).ythenx;
548
549           ! connect pullups to vdd;
550           wire ("propp.drn","vx",diff);
551           wire ("killp.drn","vddiffin",diff);
552
553           ! kill and propogate controls;
554           wire ("pffin","pffgt.in",poly);
555           wire ("pffgt.out","pffout",poly).path.y(tfbar-4).dx(1).dy(8).xtheny;
556           wire ("pttin","pttgt.in",poly).path.y(pxbar-4).dx(2).dy(8).dx(-1).
557           y(ttbar-4).xtheny;
558           wire ("pttgt.out","pttout",poly);
559           wire ("kttin","kttgt.in",poly).path.y(tfbar-6).dxy(-1,1).y(ttbar-4).
560           xtheny;
561           wire ("kttgt.out","kttout",poly).path.dy(?).xtheny;
562           wire ("ktfin","ktfgt.in",poly).path.y(tfbar-4).dxy(-1,1).ythenx;
563           wire ("ktfgt.out","ktfout",poly).path.dx(?).y(ftbar).xtheny;
564           wire ("ptfin","ptfgt.in",poly).path.y(pxbar-4).xtheny;
565           wire ("ptfgt.out","ptfout",poly).path.dy(?).xtheny;
566           wire ("pftin","pftgt.in",poly);
567           wire ("pftgt.out","pftout",poly);
568           wire ("kftin","kftgt.in",poly);
569           wire ("kftgt.out","kftout",poly).path.y(ftbar-4).dx(2).dy(8).xtheny;
570           wire ("kffin","kffgt.in",poly);
571           wire ("kffgt.out","kffout",poly);
572
```

```
573         !*************************************************************;
574         !       identify contacts                                     ;
575         !*************************************************************;
576
577         contact ("vx"); contact ("gx"); contact ("ax"); contact ("bx");
578         contact ("abarx"); contact ("bbarx");
579         contact ("ftx1"); contact ("ftx2"); contact ("ttx1");
580         contact ("ttx2"); contact ("tfx1"); contact ("tfx2");
581         contact ("ffx1"); contact ("ffx2");
582         contact ("kx1"); contact ("kx2"); contact ("px1"); contact ("px2");
583         contact ("killx").orientation (0,1);
584         contact ("propx").orientation (0,-1);
585
586         !*************************************************************;
587         !          place coordinodes                                  ;
588         !*************************************************************;
589
590         ! data input permutations - contacts;
591         place ("bin",NEW point (0,ftbar));
592         place ("bx",NEW point (pff-15,ftbar));
593         place ("ain",NEW point (0,ttbar));
594         place ("ax",NEW point (pff-12,ttbar));
595         place ("abarin",NEW point (0,pxbar+8));
596         place ("abarx",NEW point (4,pxbar+8));
597         place ("bbarin",NEW point (0,pxbar+1));
598         place ("bbarx",NEW point (4,pxbar+1));
599
600         ! data input - ft gates;
601         place ("abarftgt.in",NEW point (5,busl-1));
602         place ("abarftgt.out",NEW point (9,busl-1));
603         place ("abarftgt.src",NEW point (6,busl-2));
604         place ("abarftgt.drn",NEW point (6,busl));
605         place ("bftgt.in",NEW point (14,busl+2));
606         place ("bftgt.out",NEW point (14,busl+6));
607         place ("bftgt.src",NEW point (13,busl+3));
608         place ("bftgt.drn",NEW point (15,busl+3));
609
610         ! data input - tt gates;
611         place ("bttgt.in",NEW point (9,ttbar-3));
612         place ("bttgt.out",NEW point (9,ttbar-7));
613         place ("bttgt.src",NEW point (8,ttbar-4));
614         place ("bttgt.drn",NEW point (10,ttbar-4));
615         place ("attgt.in",NEW point (13,ttbar-3));
616         place ("attgt.out",NEW point (13,ttbar-7));
617         place ("attgt.src",NEW point (12,ttbar-4));
618         place ("attgt.drn",NEW point (14,ttbar-4));
619
620         ! data input - tf gates;
621         place ("atfgt.in",NEW point (13,ttbar-10));
622         place ("atfgt.out",NEW point (13,ttbar-14));
623         place ("atfgt.src",NEW point (12,ttbar-12));
624         place ("atfgt.drn",NEW point (14,ttbar-12));
```

```
625             place ("bbartfgt.in",NEW point (17,ttbar-14));
626             place ("bbartfgt.out",NEW point (17,ttbar-10));
627             place ("bbartfgt.src",NEW point (16,ttbar-12));
628             place ("bbartfgt.drn",NEW point (18,ttbar-12));
629
630             ! data input - ff gates;
631             place ("abarffgt.in",NEW point (8,pxbar+7));
632             place ("abarffgt.out",NEW point (12,pxbar+7));
633             place ("abarffgt.src",NEW point (9,pxbar+8));
634             place ("abarffgt.drn",NEW point (9,pxbar+6));
635             place ("bbarffgt.in",NEW point (8,pxbar+2));
636             place ("bbarffgt.out",NEW point (12,pxbar+2));
637             place ("bbarffgt.src",NEW point (9,pxbar+3));
638             place ("bbarffgt.drn",NEW point (9,pxbar+1));
639
640             ! horizontal lines to carry permutations to controls;
641             place ("ftx1",NEW point (pff-7,ftbar));
642             place ("ftx2",NEW point (pft+4,ftbar));
643             place ("ttx1",NEW point (pff-4,ttbar));
644             place ("ttx2",NEW point (ptt+4,ttbar));
645             place ("tfx1",NEW point (pff-3,tfbar));
646             place ("tfx2",NEW point (ktf+3,tfbar));
647             place ("ffx1",NEW point (pff-4,pxbar));
648             place ("ffx2",NEW point (kff+4,pxbar+10));
649
650             ! vertical control line for propogate and kill;
651
652             ! propogate - ff;
653             place ("pffin",NEW point (pff,0));
654             place ("pffgt.in",NEW point (pff,pxbar-2));
655             place ("pffgt.out",NEW point (pff,pxbar+2));
656             place ("pffgt.src",NEW point (pff-1,pxbar));
657             place ("pffgt.drn",NEW point (pff+1,pxbar));
658             place ("pffout",NEW point (pff,ylim));
659
660             ! propogate - tt;
661             place ("pttin",NEW point (ptt,0));
662             place ("pttgt.in",NEW point (ptt,ttbar-2));
663             place ("pttgt.out",NEW point (ptt,ttbar+2));
664             place ("pttgt.src",NEW point (ptt+1,ttbar));
665             place ("pttgt.drn",NEW point (ptt-1,ttbar));
666             place ("pttout",NEW point (ptt,ylim));
667
668             ! kill - tt;
669             place ("kttin",NEW point (ktt,0));
670             place ("kttgt.in",NEW point (ktt+2,ttbar-2));
671             place ("kttgt.out",NEW point (ktt+2,ttbar+2));
672             place ("kttgt.src",NEW point (ktt+1,ttbar));
673             place ("kttgt.drn",NEW point (ktt+3,ttbar));
674             place ("kttout",NEW point (ktt,ylim));
675
676             ! kill - tf;
```

```
677             place ("ktfin",NEW point (ktf,0));
678             place ("ktfgt.in",NEW point (ktf+1,tfbar+6));
679             place ("ktfgt.out",NEW point (ktf+5,tfbar+6));
680             place ("ktfgt.src",NEW point (ktf+3,tfbar+5));
681             place ("ktfgt.drn",NEW point (ktf+2,tfbar+7));
682             place ("ktfout",NEW point (ktf,ylim));
683
684             ! propogate - tf;
685             place ("ptfin",NEW point (ptf,0));
686             place ("ptfgt.in",NEW point (ptf-5,pxbar-2));
687             place ("ptfgt.out",NEW point (ptf-5,pxbar+2));
688             place ("ptfgt.src",NEW point (ptf-4,pxbar));
689             place ("ptfgt.drn",NEW point (ptf-6,pxbar));
690             place ("ptfout",NEW point (ptf,ylim));
691
692             ! propogate - ft;
693             place ("pftin",NEW point (pft,0));
694             place ("pftgt.in",NEW point (pft,pxbar-2));
695             place ("pftgt.out",NEW point (pft,pxbar+2));
696             place ("pftgt.src",NEW point (pft+1,pxbar));
697             place ("pftgt.drn",NEW point (pft-1,pxbar));
698             place ("pftout",NEW point (pft,ylim));
699
700             ! kill - ft;
701             place ("kftin",NEW point (kft,0));
702             place ("kftgt.in",NEW point (kft,ttbar-2));
703             place ("kftgt.out",NEW point (kft,ttbar+2));
704             place ("kftgt.src",NEW point (kft-1,ttbar));
705             place ("kftgt.drn",NEW point (kft+1,ttbar));
706             place ("kftout",NEW point (kft,ylim));
707
708             ! kill - ff;
709             place ("kffin",NEW point (kff,0));
710             place ("kffgt.in",NEW point (kff,pxbar+8));
711             place ("kffgt.out",NEW point (kff,pxbar+12));
712             place ("kffgt.src",NEW point (kff+1,pxbar+10));
713             place ("kffgt.drn",NEW point (kff-1,pxbar+10));
714             place ("kffout",NEW point (kff,ylim));
715
716             ! gather outcomes together and invert;
717
718             ! kill;
719             place ("ftfork",NEW point (pft+3,ttbar));
720             place ("ftfork",NEW point (pft+3,ttbar));
721             place ("kx1",NEW point (kff-4,ttbar));
722             place ("kx2",NEW point (ktf+2,ttbar));
723             place ("killx",NEW point (kff+5,ttbar));
724             place ("killp.src",NEW point (kff+6,ttbar));
725             place ("killp.drn",NEW point (kff+6,ttbar+12));
726             place ("vddiffin",NEW point (kff+6,ylim));
727             place ("vddiffout",NEW point (kff+6,0));
728
```

```
729             ! propogate;
730             place ("px1",NEW point (ptf-1,pxbar));
731             place ("px2",NEW point (ptt-2,pxbar));
732             place ("propx",NEW point (kff+5,pxbar+2));
733             place ("propp.src",NEW point (kff+6,pxbar+2));
734             place ("propp.drn",NEW point (kff+6,pxbar-10));
735             place ("vx",NEW point (kff+6,vdd));
736
737             ! propogate and kill outputs;
738             place ("kbar",NEW point (xlim,gnd+3));
739             place ("pbar",NEW point (xlim,gnd-3));
740
741             ! power and ground;
742             place ("vddin",NEW point (0,vdd));
743             place ("vddout",NEW point (xlim,vdd));
744             place ("gndin",NEW point (0,gnd));
745             place ("gx",NEW point (7,ttbar-12));
746             place ("gndout",NEW point (xlim,gnd));
747
748             ! data busses;
749             place ("inbus1",NEW point (0,bus1));
750             place ("outbus1",NEW point (xlim,bus1));
751             place ("inbus2",NEW point (0,bus2));
752             place ("outbus2",NEW point (xlim,bus2));
753
E20          754         END of aluinputdef;
755
756         !************************************************************;
757         !              alu carry chain block                         ;
758         !************************************************************;
759
760         blockdef CLASS alucarrychaindef
761         (xlim,carx,prex,amp,type);
762         VALUE type,amp; TEXT type; BOOLEAN amp; REAL xlim,carx,prex;
B21          763         BEGIN
764            REAL kpgndx,cary,prey;
765
766            ! set up parameter defaults;
767            default (xlim,51);   default (carx,22);   default (prex,32);
768
769            ! set up derived baselines;
770            kpgndx:=carx-10; cary:=bus2+7; prey:=bus1-7;
771
772            nameblock (conc ("carry.",type));
773            ! box is actually a polygon;
774            blockboundbox (0,0,xlim,ylim);
775
776            !*********************************************************;
777            ! external pin specification;
778            !*********************************************************;
779
780            ! power and ground connections;
```

```
781         pin ("vddin"); pin ("vddout"); pin ("gndin"); pin ("gndout");
782         pin ("vddiffin1"); pin ("vddiffin2"); pin ("vddiffin3");
783         pin ("vddiffout1"); pin ("vddiffout2"); pin ("vddiffout3");
784
785         ! bus connections;
786         pin ("inbus1"); pin ("outbus1"); pin ("inbus2"); pin ("outbus2");
787
788         ! carry inputs and outputs and propogate output;
789         pin ("cin"); pin ("cout"); pin ("pout"); pin ("pbarout");
790         pin ("cbarout"); pin ("pbarout"); pin ("carout");
791
792         ! kill and propogate inputs;
793         pin ("kbar"); pin ("pbar");
794
795         ! precharge input and output;
796         pin ("prein"); pin ("preout");
797
798         !*************************************************************;
799         !       transistor components                                 ;
800         !*************************************************************;
801
802         ! propogate and carry-kill input circuits;
803         pulldown ("kbargt"); pullup ("kbarpull","kbarx");
804         passtran ("kpregt"); passtran ("kgt").intcout;
805         pulldown ("pbargt"); pullup ("pbarpull","pbarx").drntosrc.ythenx;
806         passtran ("pgt").intoout; passtran ("cpregt");
807
808         ! carry in inversion cicuit;
809         pulldown ("cgt"); pullup ("cpull","cx1").drntosrc.xtheny;
810         pulldown ("cbargt"); pullup ("cbarpull","cx2");
811
812         ! propogate inverter output;
813         pullup ("pinvpull","pinvx"); pulldown ("pinvgt");
814
815         !*************************************************************;
816         !                       wires                                 ;
817         !*************************************************************;
818
819         ! connect up propogate and carry-kill components;
820         wire ("kbar","kbargt.in",poly);
821         wire ("kbargt.src","gx1",diff).path.ythenx;
822         wire ("kbargt.drn","kbarx",diff);
823         wire ("kbarpull.drn","vddiffin1",diff);
824         wire ("pbar","pbargt.in",poly);
825         wire ("pbargt.src","gx1",diff).path.ythenx;
826         wire ("pbargt.drn","pbarx",diff).path.dy(-2).x(carx-8).ythenx;
827         wire ("pbarpull.drn","vx1",diff);
828         wire ("vx1","vddiffout1",diff).path.xtheny;
829         wire ("kbarx","kgt.in",poly).path.ythenx;
830         wire ("kgt.src","gkfork",diff).path.dy(-2).xtheny;
831         wire ("kgt.drn","kpfork",diff).path.ythenx;
832         wire ("pgt.drn","kpfork",diff);
```

```
833             wire ("gx1","gkfork",diff).path.dy(1).xtheny;
834             wire
835             ("pbarx","pgt.in",poly).path.dx(-1).dy(-4).dx(6).y(gnd-2).dx(2).ythenx;
836
837             ! connect up pre-charge;
838             wire ("prex","prein",poly).path.dx(-1).y(bus2-2.5).xtheny;
839             wire ("prex","kprex",metal).path.ythenx;
840             wire ("kprex","kpregt.in",poly);
841             wire ("kpregt.src","gkfork",diff);
842             wire ("kpregt.drn","kbarx",diff).path.xtheny;
843             wire ("prex","cprex", metal).path.ythenx;
844             wire ("cprex","carfork",poly).path.ytherx;
845             wire ("carfork","cpregt.in",poly);
846             wire ("vddiffin2","cpregt.drn",diff);
847             wire ("vx2","vddiffout2",diff).path.xtheny;
848             wire ("cpregt.src","pgt.drn",diff).path.ythenx;
849             wire ("carfork","preout",poly);
850
851             ! connect up carry-in to output to next stage;
852             wire ("cin","cinx1",diff).path.ythenx;
853             wire ("cinx3","pgt.src",diff).path.dx(1).y(prey+3).xtheny;
854             IF amp THEN wire ("cinx3","cx2",metal).path.ythenx
855             ELSE wire ("cinx1","cinx3",diff).path.dx(1).ythenx;
856             wire ("cinx1","cinx2",metal).path.ythenx;
857             wire ("cinx2","cgt.in",poly).path.xtheny;
858             wire ("cgt.src","cinfork",diff).path.xtheny;
859             wire ("cinfork","gx2",diff).path.ythenx;
860             wire ("cx1","cgt.drn",diff).path.dy(2).dx(-5).ythenx;
861             wire ("cx1","cbargt.in",poly).path.dy(-1).dx(3).dy(6).xtheny;
862             wire ("cpull.drn","vx3",diff).path.xthery;
863             wire ("cbargt.src","cinfork",diff);
864             wire ("cbargt.drn","cx2",diff).path.xtheny;
865             wire ("cbarpull.drn","vx3",diff);
866
867             ! connect carry-in and carray-out bar to outputs;
868             wire ("cx1","cbarout",metal).path.ythenx;
869             wire ("cx2","carout",poly).path.dy(-2).dx(3).ythenx;
870
871             ! connect up carry-out;
872             wire ("cout","kpfork",diff);
873
874             ! connect up prograte inverter and outputs;
875             wire ("vddiffin3","pinvpull.drn",diff);
876             wire ("pinvx","pbarout",poly).path.ythenx;
877             wire ("pinvx","pinvgt.drn",diff).path.dy(-4).xtheny;
878             wire ("pinvgt.src","gx2",diff).path.dy(-3).xtheny;
879             wire ("pgt.out","pinvgt.in",poly).path.dx(2).ythenx;
880             wire ("pinvgt.out","pout",poly);
881             wire ("vx3","vddiffout3",diff).path.dy(-1).xtheny;
882
883             ! busses connect across cell;
884             wire ("inbus1","outbus1",metal);
```

```
885           wire ("inbus2","outbus2",metal);
886
887           ! connect up power and ground across block;
888           wire ("vddin","vx1",metal).width (4);
889           wire ("vx1","vx2",metal).width (4);
890           wire ("vx2","vx3",metal).width(4);
891           wire ("vx3","vddout",metal).width (4);
892           wire ("gndin","gx1",metal).width (4);
893           wire ("gx1","gx2",metal).width (4);
894           wire ("gx2","gndout",metal).width (4);
895
896           !************************************************************;
897           !                    contacts                                ;
898           !************************************************************;
899
900           ! power and ground;
901           contact ("vx1"); contact ("vx2"); contact ("vx3");
902           contact ("gx1"); contact ("gx2");
903
904           ! precharge contacts;
905           contact ("prex"); contact ("kprex"); contact ("cprex");
906
907           ! carry-in contacts;
908           contact ("cinx1"); contact ("cinx2"); contact ("cinx3");
909           contact ("cx1").oriertation (0,-1);
910           contact ("cx2").oriertation (0,-1);
911
912           ! contacts for kill and propogate pullups;
913           contact ("pbarx").orientation (-1,0);
914           contact ("kbarx").orientation (0,1);
915           contact ("pinvx").orientation (0,1);
916
917           !************************************************************;
918           !       place coordirodes relative to parameters           ;
919           !************************************************************;
920
921           ! power and ground placements;
922           place ("vddin",NEW point (0,vdd));
923           place ("vddout",NEW point (xlim,vdd));
924           place ("gndin",NEW point (0,gnd));
925           place ("gndout",NEW point (xlim,gnd));
926           place ("vddiffin1",NEW point (3,ylim));
927           place ("vddiffin2",NEW point (prex-4,ylim));
928           place ("vddiffin3",NEW point (prex+11,ylim));
929           place ("vddiffout1",NEW point (3,0));
930           place ("vddiffout2",NEW point (carx+6,0));
931           place ("vddiffout3",NEW point (prex+11,0));
932           place ("gx1",NEW point (carx-10,gnd));
933           place ("gx2",NEW point (carx+7,gnd));
934           place ("vx1",NEW point (carx-15,vdd));
935           place ("vx2",NEW point (carx+6,vdd));
936           place ("vx3",NEW point (prex+15,vdd));
```

```
937
938           ! bus input/output placements;
939           place ("inbus1",NEW point (0,bus1));
940           place ("outbus1",NEW point (xlim,bus1));
941           place ("inbus2",NEW point (0,bus2));
942           place ("outbus2",NEW point (xlim,bus2));
943
944           ! place carry inputs and outputs and propogate output;
945           place ("cin",NEW point (carx,0));
946           place ("cout",NEW point (carx,ylim));
947           place ("pout",NEW point (xlim,prey-7.5));
948           place ("cbarout",NEW point (xlim,cary-1));
949           place ("carout",NEW point (xlim,cary+9));
950           place ("pbarout",NEW point (xlim,bus1-6.5));
951
952           ! place kill and propogate inputs;
953           place ("kbar",NEW point (0,gnd+3));
954           place ("pbar",NEW point (0,gnd-3));
955           place ("kbarx",NEW point (3,prey));
956           place ("pbarx",NEW point (carx-10,cary));
957
958           ! place precharge input and output;
959           place ("prein",NEW point (prex,0));
960           place ("preout",NEW point (prex,ylim));
961
962           ! propogate and carry-kill inverter pulldowns;
963           place ("kbargt.in",position ("kbar"));
964           place ("kbargt.out",NEW point (4,gnd+3));
965           place ("kbargt.src",NEW point (3,gnd+2));
966           place ("kbargt.drn",NEW point (3,gnd+4));
967           place ("pbargt.in",position ("pbar"));
968           place ("pbargt.out",NEW point (4,gnd-3));
969           place ("pbargt.src",NEW point (3,gnd-2));
970           place ("pbargt.drn",NEW point (3,gnd-4));
971
972           ! propogate and carry-kill inverter pullups;
973           place ("kbarpull.src",NEW point (3,prey));
974           place ("kbarpull.drn",NEW point (3,prey+9));
975           place ("pbarpull.src",NEW point (carx-10,cary));
976           place ("pbarpull.drn",NEW point (carx-15,cary-5));
977
978           ! place propogate inverter pullup and pulldown;
979           place ("pinvpull.src",NEW point (prex+11,bus1-7.5));
980           place ("pinvpull.drn",NEW point (prex+11,bus1+0.5));
981           place ("pinvx",NEW point (prex+11,bus1-7.5));
982           place ("pinvgt.in",NEW point (prex+2,prey-7.5));
983           place ("pinvgt.out",NEW point (prex+4,prey-7.5));
984           place ("pinvgt.src",NEW point (prex+3,prey-8.5));
985           place ("pinvgt.drn",NEW point (prex+3,prey-6.5));
986
987           ! generating carry out pass transistors;
988           place ("pgt.in",NEW point (carx-1,bus1-2.5));
```

```
        989             place ("pgt.out",NEW point (carx+7,bus1-2.5));
        990             place ("pgt.src",NEW point (carx,bus1-3.5));
        991             place ("pgt.drn",NEW point (carx,bus1-1.5));
        992             place ("kgt.in",NEW point (carx-12,bus1-2.5));
        993             place ("kgt.out",NEW point (carx-8,bus1-2.5));
        994             place ("kgt.src",NEW point (carx-10,bus1-3.5));
        995             place ("kgt.drn",NEW point (carx-10,bus1-1.5));
        996             place ("kpfork",NEW point (carx,bus1+0.5));
        997
        998             ! precharge components;
        999             place ("prex",NEW point (carx+4,prey-2));
       1000             place ("kprex",NEW point (carx-10,prey-3));
       1001             place ("cprex",NEW point (prex+4,prey));
       1002             place ("kpregt.in",NEW point (kpgndx,prey-6));
       1003             place ("kpregt.out",NEW point (kpgndx,prey-8));
       1004             place ("kpregt.src",NEW point (kpgndx+1,prey-7));
       1005             place ("kpregt.drn",NEW point (kpgndx-1,prey-7));
       1006             place ("gkfork",NEW point (carx-6,prey-7));
       1007             place ("cpregt.in",NEW point (prex-2,bus1+2.5));
       1008             place ("cpregt.out",NEW point (prex-6,bus1+2.5));
       1009             place ("cpregt.src",NEW point (prex-4,bus1+1.5));
       1010             place ("cpregt.drn",NEW point (prex-4,bus1+3.5));
       1011             place ("carfork",NEW point (prex,bus1+2.5));
       1012
       1013             ! carry in connection to output stage;
       1014             place ("cinx1",NEW point (carx-1,cary-0.5));
       1015             place ("cinx2",NEW point (prex,cary+0.5));
       1016             place ("cinx3",NEW point (carx-1,cary+7.5));
       1017             place ("cgt.in",NEW point (prex-1,cary-2.5));
       1018             place ("cgt.out",NEW point (prex-1,cary-4.5));
       1019             place ("cgt.src",NEW point (prex-2,cary-3.5));
       1020             place ("cgt.drn",NEW point (prex,cary-3.5));
       1021             place ("cx1",NEW point (prex+9,cary+1));
       1022             place ("cpull.src",NEW point (prex+9,cary+1));
       1023             place ("cpull.drn",NEW point (prex+12,bus2-4.5));
       1024             place ("cbargt.in",NEW point (prex+7,cary+8));
       1025             place ("cbargt.out",NEW point (prex+7,cary+20));
       1026             place ("cbargt.src",NEW point (prex+6,cary+15));
       1027             place ("cbargt.drn",NEW point (prex+8,cary+15));
       1028             place ("cx2",NEW point (prex+15,cary+13));
       1029             place ("cbarpull.src",position ("cx2"));
       1030             place ("cbarpull.drn",NEW point (prex+15,cary+9));
       1031             place ("cinfork",NEW point (carx+6,cary+15));
       1032
E21    1033         END;
       1034
       1035         !*****************************************************************;
       1036         !                        alu output block                       ;
       1037         !*****************************************************************;
       1038
       1039         blockdef CLASS aluoutputdef  (xlim,rst,rinc,latch,dr1,dr2,type);
       1040         VALUE type; TEXT type; REAL xlim,rst,rinc,latch,dr1,dr2;
```

```
B22     1041        BEGIN
        1042           REAL rxbar,routbar,tfbar,ftbar,ttbar,cinbl,cinx,pinvx,din2,
        1043           ttin,ffin,tfin,ftin;
        1044
        1045           nameblock (conc ("output.",type));
        1046
        1047           ! set up default parameter settings;
        1048           default (xlim,90);   default (rst,39);   default (rinc,7);
        1049           default (latch,73);   default (dr1,81);   default (dr2,87);
        1050
        1051           ! set up derived baselines;
        1052           rxbar:=bus2+13; tfbar:=routbar:=bus1-8.5;  ftbar:=tfbar-7; cinbl:=8;
        1053           cinx:=16; pinvx:=bus1-4; din2:=latch-6; ffin:=rst+2; ttbar:=rxbar+8;
        1054           tfin:=rst+rinc+1; ftin:=rst+2*rinc; ttin:=rst+3*rinc-1;
        1055
        1056           ! set up the bounding box of the block's physical description;
        1057           blockboundbox (0,0,xlim,ylim);
        1058
        1059           !*************************************************************;
        1060           !              specify external interface                   ;
        1061           !*************************************************************;
        1062
        1063           ! power and ground connections;
        1064           pin ("vddin"); pin ("vddout"); pin ("gndin"); pin ("gndout");
        1065
        1066           ! data busses;
        1067           pin ("inbus1"); pin ("outbus1"); pin ("inbus2"); pin ("outbus2");
        1068
        1069           ! propogate and carry inputs;
        1070           pin ("propin");  pin ("pbarin");
        1071           pin ("cin"); pin ("cinbar");
        1072
        1073           ! output function pins ;
        1074           pin ("rffin"); pin ("rffout"); pin ("rtfin"); pin ("rtfout");
        1075           pin ("rftin"); pin ("rftout"); pin ("rttin"); pin ("rttout");
        1076
        1077           ! latch and bus drive control pins;
        1078           pin ("latchin"); pin ("latchout"); pin ('drivelin");
        1079           pin ("drivelout"); pin ("drive2in"); pin ("drive2out");
        1080
        1081           !*************************************************************;
        1082           !              main components                              ;
        1083           !*************************************************************;
        1084
        1085           ! permute propogate and carry in signals;
        1086           pulldown ("cinbartfgt"); pulldown ("ptfgt"); pulldown ("cinftgt");
        1087           pulldown ("pbarftgt"); pulldown ("pttgt"); pulldown ("cinttgt");
        1088           pulldown ("cinbarffgt"); pulldown ("pbarffgt");
        1089
        1090           ! result controls;
        1091           pulldown ("rffgt"); pulldown ("rtfgt");
        1092           pulldown ("rftgt"); pulldown ("rttgt");
```

```
1093
1094            ! latch and drive controls;
1095            passtran ("latchgt"); passtran ("rbargt"); passtran ("drivelgt");
1096            passtran ("drive2gt").intoout.dxy(1,1).dy(6).dxy(1,1);
1097
1098            ! pullup for result ;
1099            pullup ("rpull","rpullx");
1100
1101            !*****************************************************************;
1102            !                connect up components and pins                   ;
1103            !*****************************************************************;
1104
1105            ! permutations of carry in and propogate  (control);
1106            wire ("pbarin","pbarftgt.in",poly).path.xtheny;
1107            wire ("pbarftgt.out","pbarffgt.in",poly).path.y(ttbar+4).xtheny;
1108            wire ("propin","ptfgt.in",poly).path.xtheny;
1109            wire ("ptfgt.out","pttgt.in",poly).path.ythenx;
1110            wire ("cbarfk","cbarx",metal);
1111            wire ("cbarx","cinbartfgt.in",poly);
1112            wire ("cinbar","cinbarx",metal).path.xtheny;
1113            wire ("cinbarx","cinbarffgt.in",poly).path.xtheny;
1114            wire ("cin","cinttgt.in",poly).path.x(cinbl+8).ythenx;
1115            wire ("cinttgt.out","cinftgt.in",poly).path.xtheny;
1116
1117            ! diffusion paths through permutation wiring;
1118            wire ("gx4","cinbartfgt.src",diff).path.ythenx;
1119            wire ("cinbartfgt.drn","ptfgt.src",diff).path.dx(2).ythenx;
1120            wire ("ptfgt.drn","tfx1",diff).path.dx(1).dxy(2,2).ythenx;
1121            wire ("gx1","cinbarffgt.src",diff).path.ythenx;
1122            wire ("cinbarffgt.drn","pbarffgt.src",diff);
1123            wire ("gx2","cinftgt.src",diff).path.ythenx;
1124            wire ("cinftgt.drn","pbarftgt.src",diff);
1125            wire ("pbarftgt.drn","ftx1",diff);
1126            wire ("gx2","pttgt.src",diff).path.xtheny;
1127            wire ("pttgt.drn","cinttgt.src",diff);
1128            wire ("cinttgt.drn","ttx1",diff).path.ythenx;
1129
1130            ! supply permutations to result controls;
1131            wire ("tfx1","tfx2",metal).path.ythenx;
1132            wire ("tfx2","rtfgt.src",diff);
1133            wire ("ftx1","ftx2",metal);
1134            wire ("ftx2","rftgt.src",diff).path.dx(-1).ythenx;
1135            wire ("ttx1","ttx2",metal).path.ythenx;
1136            wire ("ttx2","rttgt.src",diff);
1137            wire ("pbarffgt.drn","rffgt.src",diff);
1138
1139            ! amalgamate result controls;
1140            wire ("rtfgt.drn","rx1",diff).path.dx(-2).ythenx;
1141            wire ("rffgt.drn","rx1",diff);
1142            wire ("rftgt.drn","rtfork",diff);
1143            wire ("rttgt.drn","rtfork",diff);
1144            wire ("rtfork","rx2",diff).path.ythenx;
```

```
1145            wire ('rx1","rpullx",metal).path.xtheny;
1146            wire ('rx2","rpullx",metal).path.xtheny;
1147
1148            ! invert result and pass through latch;
1149            wire ('vx","rpull.drn",diff);
1150            wire ('rpullx","latchgt.drn",diff).path.dy(2).xtheny;
1151            wire ('latchgt.src","rbarx",diff).path.dy(2).x(ttin+4).ythenx;
1152            wire ('rbarx","rbargt.in",poly).path.xtheny;
1153
1154            ! connect up output to busses;
1155            wire ("gx3","rbargt.src",diff);
1156            wire ('rbargt.drn","routx1",diff);
1157            wire ("routx1","routx2",metal);
1158            wire ("routx2","drivelgt.src",diff).path.dx(1).ythenx;
1159            wire ("routx2","drive2gt.src",diff).path.dx(1).ythenx;
1160            wire ("drivelgt.drn","bus1x",diff);
1161            wire ("drive2gt.drn","bus2x",diff).path.dx(1).ythenx;
1162
1163            ! control paths;
1164            wire ("rffin","rffgt.in",poly);
1165            wire ("rffgt.out","rffout",poly);
1166            wire ("rtfin","rtfgt.in",poly).path.y(rxbar-4).dx(2).dy(8).xtheny;
1167            wire ("rtfgt.out","rtfout",poly);
1168            wire ("rftin","rftgt.in",poly);
1169            wire
1170            ("rftgt.out","rftout",poly).path.y(ftbar-4).dx(2).y(tfbar+4).xtheny;
1171            wire ("rttin","rttgt.in",poly).path.y(rxbar-4).dx(2).dy(8).xtheny;
1172            wire ("rttgt.out","rttout",poly);
1173            wire ("latchin","latchgt.in",poly).path.y(rxbar-2).dx(2).ythenx;
1174            wire ("latchgt.in","latchout",poly).path.dx(3).y(bus1-4).xtheny;
1175            wire ("drivelin","fix",poly).path.y(bus2-8).dx(-1).dy(12).xtheny;
1176            wire ("fix","drivelgt.in",poly).path.y(routbar-4).dx(-2).dy(8).xtheny;
1177            wire ("drivelgt.out","drivelout",poly);
1178            wire
1179            ("drive2in","drive2gt.in",poly).path.y(bus2-4).dx(-4).dy(5).dxy(1,1);
1180            wire ("drive2gt.out","drive2out",poly).path.dxy(1,1).y(ylim);
1181
1182            ! power ground and busses;
1183            wire ("vddin","vx",metal).width (4);
1184            wire ("vx","vddout",metal).width (4);
1185            wire ("gndin","gx4",metal);
1186            wire ("gx4","gx1fork",metal).width (4);
1187            wire ("gx1fork","gx2fork",metal).width (4);
1188            wire ("gx2fork","gx3fork",metal).width (4);
1189            wire ("gx3fork","gndout",metal).width (4);
1190            wire ("gx1fork","gx1",metal).path.ythenx;
1191            wire ("gx2fork","gx2",metal);
1192            wire ("gx3fork","gx3",metal).path.ythenx;
1193            wire ("inbus1","bus1x",metal);
1194            wire ("bus1x","outbus1",metal);
1195            wire ("inbus2","bus2x",metal);
1196            wire ("bus2x","outbus2",metal);
```

```
1197
1198            !*********************************************************;
1199            !                    contacts                             ;
1200            !*********************************************************;
1201
1202            contact ("tfx1"); contact ("tfx2"); contact ("ftx1");
1203            contact ("ftx2"); contact ("ttx1"); contact ("ttx2");
1204            contact ("rx1"); contact ("rx2"); contact ("vx"); contact ("gx1");
1205            contact ("gx2"); contact ("gx3"); contact ("gx4"); contact ("busIx");
1206            contact ("bus2x"); contact ("routx1"); contact ("routx2");
1207            contact ("cinbarx"); contact ("cbarx");
1208            contact ("rbarx").orientation (1,0);
1209            contact ("rpullx").orientation (0,-1);
1210
1211            !*************************************************************;
1212            !              place coordinodes                               ;
1213            !*************************************************************;
1214
1215            ! propogate ;
1216            place ("propin",NEW point (0,bus1-14.5));
1217            place ("pbarin",NEW point (0,bus1-6.5));
1218
1219            ! carry in and propogate and carry in bar 1;
1220            place ("cinbartfgt.in",NEW point (cinb1,gnd-1));
1221            place ("cinbartfgt.out",NEW point (cinb1,gnd+3));
1222            place ("cinbartfgt.src",NEW point (cinb1-1,gnd+1));
1223            place ("cinbartfgt.drn",NEW point (cinb1+1,gnd+1));
1224            place ("ptfgt.src",NEW point (cinb1+6,gnd+4));
1225            place ("ptfgt.drn",NEW point (cinb1+8,gnd+4));
1226            place ("ptfgt.in",NEW point (cinb1+7,gnd+6));
1227            place ("ptfgt.out",NEW point (cinb1+7,gnd+2));
1228            place ("cin",NEW point (0,rxbar+3));
1229            place ("cinttgt.in",NEW point (cinx+4,gnd-4));
1230            place ("cinttgt.out",NEW point (cinx+8,gnd-4));
1231            place ("cinttgt.src",NEW point (cinx+7,gnd-3));
1232            place ("cinttgt.drn",NEW point (cinx+7,gnd-5));
1233            place ("pttgt.in",NEW point (cinx+4,gnd));
1234            place ("pttgt.out",NEW point (cinx+8,gnd));
1235            place ("pttgt.src",NEW point (cinx+7,gnd+1));
1236            place ("pttgt.drn",NEW point (cinx+7,gnd-1));
1237
1238            ! carry in bar 2;
1239            place ("cinbar",NEW point (0,rxbar-7));
1240            place ("cinbarx",NEW point (rst-7,rxbar-6));
1241            place ("cbarfk",NEW point (cinb1,rxbar-7));
1242            place ("cbarx",NEW point (cinb1,gnd-7));
1243            place ("cinbarffgt.in",NEW point (rst-6,rxbar-3));
1244            place ("cinbarffgt.out",NEW point (rst-6,rxbar+1));
1245            place ("cinbarffgt.src",NEW point (rst-7,rxbar));
1246            place ("cinbarffgt.drn",NEW point (rst-5,rxbar));
1247            place ("pbarffgt.in",NEW point (rst-2,rxbar+1));
1248            place ("pbarffgt.out",NEW point (rst-2,rxbar-3));
```

```
1249            place ("pbarffgt.src",NEW point (rst-3,rxbar));
1250            place ("pbarffgt.drn",NEW point (rst-1,rxbar));
1251
1252            ! carry in and propogate inverse;
1253            place ("cinftgt.in",NEW point (rst-10,fttar-2));
1254            place ("cinftgt.out",NEW point (rst-10,fttar+2));
1255            place ("cinftgt.src",NEW point (rst-11,fttar));
1256            place ("cinftgt.drn",NEW point (rst-9,fttar));
1257            place ("pbarftgt.in",NEW point (rst-6,fttar+2));
1258            place ("pbarftgt.out",NEW point (rst-6,fttar-2));
1259            place ("pbarftgt.src",NEW point (rst-7,fttar));
1260            place ("pbarftgt.drn",NEW point (rst-5,fttar));
1261
1262            ! result controls;
1263            place ("rffin",NEW point (rst+2,0));
1264            place ("rffgt.in",NEW point (rst+2,rxbar-3));
1265            place ("rffgt.out",NEW point (rst+2,rxbar+1));
1266            place ("rffgt.src",NEW point (rst+1,rxbar));
1267            place ("rffgt.drn",NEW point (rst+3,rxbar));
1268            place ("rffout",NEW point (rst+2,ylim));
1269
1270            ! result controls - tf;
1271            place ("rtfin",NEW point (tfin,0));
1272            place ("rtfgt.in",NEW point (tfin,tfbar-2));
1273            place ("rtfgt.out",NEW point (tfin,tfbar+2));
1274            place ("rtfgt.src",NEW point (tfin+1,tfbar));
1275            place ("rtfgt.drn",NEW point (tfin-1,tfbar));
1276            place ("rtfout",NEW point (tfin,ylim));
1277
1278            ! result controls - ft;
1279            place ("rftin",NEW point (ftin,0));
1280            place ("rftgt.in",NEW point (ftin,ttbar-2));
1281            place ("rftgt.out",NEW point (ftin,ttbar+2));
1282            place ("rftgt.src",NEW point (ftin-1,ttbar));
1283            place ("rftgt.drn",NEW point (ftin+1,ttbar));
1284            place ("rftout",NEW point (ftin,ylim));
1285
1286            ! result controls - tt;
1287            place ("rttin",NEW point (ttin,0));
1288            place ("rttgt.in",NEW point (ttin,ttbar-2));
1289            place ("rttgt.out",NEW point (ttin,ttbar+2));
1290            place ("rttgt.src",NEW point (ttin+1,ttbar));
1291            place ("rttgt.drn",NEW point (ttin-1,ttbar));
1292            place ("rttout",NEW point (ttin,ylim));
1293
1294            ! result inverter pull up;
1295            place ("rpull.drn",NEW point (din2,rxbar-15));
1296            place ("rpull.src",NEW point (din2,rxbar-1));
1297            place ("rpullx",NEW point (din2,rxbar-1));
1298
1299            ! latch controls;
1300            place ("latchin",NEW point (latch,0));
```

```
1301            place ("latchgt.in",NEW point (latch-1,gnd));
1302            place ("latchgt.out",NEW point (latch-3,gnd));
1303            place ("latchgt.src",NEW point (latch-2,gnd+1));
1304            place ("latchgt.drn",NEW point (latch-2,gnd-1));
1305            place ("latchout",NEW point (latch,ylim));
1306
1307            ! drive bus 1 controls;
1308            place ("drivelin",NEW point (dr1,0));
1309            place ("fix",NEW point (dr1,gnd));
1310            place ("drivelgt.in",NEW point (dr1,bus1-2.5));
1311            place ("drivelgt.out",NEW point (dr1,bus1+3.5));
1312            place ("drivelgt.src",NEW point (dr1+1,bus1));
1313            place ("drivelgt.drn",NEW point (dr1-1,bus1));
1314            place ("drivelout",NEW point (dr1,ylim));
1315
1316            ! drive bus 2 controls;
1317            place ("drive2in",NEW point (dr2+1,0));
1318            place ("drive2gt.in",NEW point (dr2-2,bus2+2));
1319            place ("drive2gt.out",NEW point (dr2,bus2+10));
1320            place ("drive2gt.src",NEW point (dr2-2,bus2+6));
1321            place ("drive2gt.drn",NEW point (dr2,bus2+5));
1322            place ("drive2out",NEW point (dr2+1,ylim));
1323
1324            ! result inversion;
1325            place ("rbarx",NEW point (ttin+6,tfbar-7));
1326            place ("rbargt.in",NEW point (ttin+8,tfbar-2));
1327            place ("rbargt.out",NEW point (ttin+8,tfbar+8));
1328            place ("rbargt.src",NEW point (ttin+7,tfbar));
1329            place ("rbargt.drn",NEW point (ttin+9,tfbar));
1330
1331            ! output to busses;
1332            place ("routx1",NEW point (ttin+12,tfbar));
1333            place ("routx2",NEW point (dr1+2,tfbar));
1334
1335            ! horizontal carry/propogate to result selection;
1336            place ("tfx1",NEW point (cinx+3,tfbar-3));
1337            place ("tfx2",NEW point (tfin+4,tfbar));
1338            place ("ftx1",NEW point (rst-2,ftbar));
1339            place ("ftx2",NEW point (tfin+4,ftbar));
1340            place ("ttx1",NEW point (rst-7,ttbar-1));
1341            place ("ttx2",NEW point (ttin+6,ttbar));
1342            place ("rtfork",NEW point (ftin+3,ttbar));
1343            place ("rx1",NEW point (rst+6,rxbar));
1344            place ("rx2",NEW point (ftin+4,rxbar));
1345
1346            ! power and ground;
1347            place ("vddin",NEW point (0,vdd));
1348            place ("vx",NEW point (din2,vdd));
1349            place ("vddout",NEW point (xlim,vdd));
1350            place ("gndin",NEW point (0,gnd));
1351            place ("gx1",NEW point (rst-14,rxbar+1));
1352            place ("gx2",NEW point (cinx+8,ftbar-3));
```

```
        1353            place ("gx3",NEW point (ttin+4,tfbar));
        1354            place ("gx4",NEW point (2,gnd));
        1355            place ("gndout",NEW point (xlim,gnd));
        1356            place ("gx1fork",NEW point (cinx+5,gnd));
        1357            place ("gx2fork",NEW point (cinx+8,gnd));
        1358            place ("gx3fork",NEW point (ttin-2,gnd));
        1359
        1360            ! busses;
        1361            place ("inbus1",NEW point (0,bus1));
        1362            place ("bus1x",NEW point (dr1-4,bus1));
        1363            place ("outbus1",NEW point (xlim,bus1));
        1364            place ("inbus2",NEW point (0,bus2));
        1365            place ("bus2x",NEW point (dr2+2,bus2));
        1366            place ("outbus2",NEW point (xlim,bus2));
        1367
B22     1368        END of aluoutputstage;
        1369
        1370        !************************************************************;
        1371        !           bit slice of 2*memory+input+carry+output         ;
        1372        !************************************************************;
        1373
        1374        blockdef CLASS om2bitslice (mem1,mem2,inp,car,outp,twomem,type);
        1375        REF (memcell) mem1,mem2; REF (aluinputdef) inp;
        1376        REF (alucarrychaindef) car;
        1377        REF (aluoutputdef) outp; TEXT type; BOOLEAN twomem;
B23     1378        BEGIN
        1379           INTEGER i,xlim;
        1380           REF (memcell) mem;
        1381
        1382           ! set text name of cell definition;
        1383           nameblock (conc ("aluslice.",type));
        1384
        1385           ! set bounding box;
        1386           xlim:=IF twomem THEN mem1.xlim+mem2.xlim ELSE
        1387           inp.xlim+car.xlim+outp.xlim;
        1388           blockboundbox (0,0,xlim,ylim);
        1389
        1390           ! pin specification;
        1391
        1392           ! horizontal;
        1393           pin ("vddin"); pin ("vddout"); pin ("gndin"); pin ("gndout");
        1394           pin ("inbus1"); pin ("outbus1"); pin ("inbus2"); pin ("outbus2");
        1395
        1396           ! vertical;
        1397
B24     1398           IF twomem THEN BEGIN
        1399              pin ("zout"); pin ("zbarout"); pin ("zoverout");
        1400              pin ("zbaroverout");
        1401              ! memory cells 1 and 2;
B25     1402              FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
        1403                 pin (si ("rdmemin1",i)); pin (si("rdmemout1",i));
        1404                 pin (si("rdmemin2",i)); pin (si("rdmemout2",i));
```

```
        1405                pin (sl("drmemin1",i)); pin (sl("drmemout1",i));
        1406                pin (sl("drmemin2",i)); pin (sl("drmemout2",i));
        1407                pin (sl("refin",i)); pin (sl("refout",i));
        1408                pin (sl("vddinvin",i)); pin (sl("vddinvout",i));
E25     1409             END;
        1410
B26 E24 1411          END ELSE BEGIN
        1412             pin ("ain"); pin ("abarin"); pin ("bin"); pin ("bbarin");
        1413
        1414             ! input cell;
        1415             pin ("pffin"); pin ("pffout"); pin ("pttin"); pin ("pttout");
        1416             pin ("kttin"); pin ("kttout"); pin ("kffin"); pin ("kffout");
        1417             pin ("ptfin"); pin ("ptfout"); pin ("pftin"); pin ("pftout");
        1418             pin ("kftin"); pin ("kftout"); pin ("kftin"); pin ("kffout");
        1419             pin ("vddinpout"); pin ("vddinpin");
        1420
        1421             ! carry cell;
        1422             pin ("vddcarout1"); pin ("vddcarin1"); pin ("vddcarout2");
        1423             pin ("vddcarin2");
        1424             pin ("vddcarout3"); pin ("vddcarin3");
        1425             pin ("cin"); pin ("cout"); pin ("prein"); pin ("preout");
        1426
        1427             ! output cell;
        1428             pin ("rffin"); pin ("rffout"); pin ("rtfin"); pin ("rtfout");
        1429             pin ("rftin"); pin ("rftout"); pin ("rttin"); pin ("rttout");
        1430             pin ("latchin"); pin ("latchout");
        1431             pin ("dr1resin"); pin ("dr1resout"); pin ("dr2resin");
        1432             pin ("dr2resout");
        1433
E26     1434          END;
        1435
B27     1436          IF twomem THEN BEGIN
        1437             ! instance 2 memory cells;
B28     1438             FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
        1439                mem:-IF i=1 THEN mem1 ELSE mem2;
        1440                blockinst (sl ("mem",i),mem,NEW trarsform (NONE).translatedby
        1441                 (NEW point ((i-1)*mem.xlim,0)));
E28     1442             END;
B29 E27 1443          END ELSE BEGIN
        1444
        1445
        1446             ! instance input carry and output;
        1447             blockinst ("input",inp,NEW transform (NONE).translatedby
        1448             (NEW point (0,0)));
        1449             blockinst ("carry",car,NEW transform (NONE).translatedby
        1450             (NEW point (inp.xlim,0)));
        1451             blockinst ("output",outp,NEW transform (NONE).translatedby
        1452             (NEW point (inp.xlim+car.xlim,0)));
E29     1453          END;
        1454
        1455          ! tie up horizontal bussing;
        1456
```

```
B30       1457          IF twomem THEN BEGIN
          1458             ! power;
          1459             wire ("vddin","mem(1).vddin",metal);
          1460             wire ("mem(1).vddout","mem(2).vddin",metal);
          1461             wire ("mem(2).vddout","vddout",metal);
B31 E30   1462          END ELSE BEGIN
          1463             wire ("vddin","input.vddin",metal);
          1464             wire ("input.vddout","carry.vddin",metal);
          1465             wire ("carry.vddout","output.vddin",metal);
          1466             wire ("output.vddout","vddout",metal);
E31       1467          END;
          1468
          1469          ! ground;
B32       1470          IF twomem THEN BEGIN
          1471             wire ("gndin","mem(1).gndin",metal);
          1472             wire ("mem(1).gndout","mem(2).gndin",metal);
          1473             wire ("mem(2).gndout","gndout",metal);
B33 E32   1474          END ELSE BEGIN
          1475             wire ("gndin","input.gndin",metal);
          1476             wire ("input.gndout","carry.gndin",metal);
          1477             wire ("carry.gndout","output.gndin",metal);
          1478             wire ("output.gndout","gndout",metal);
E33       1479          END;
          1480
          1481          ! bus 1;
B34       1482          IF twomem THEN BEGIN
          1483             wire ("inbus1","mem(1).inbus1",metal);
          1484             wire ("mem(1).outbus1","mem(2).inbus1",metal);
          1485             wire ("mem(2).outbus1","outbus1",metal);
B35 E34   1486          END ELSE BEGIN
          1487             wire ("inbus1","input.inbus1",metal);
          1488             wire ("input.outbus1","carry.inbus1",metal);
          1489             wire ("carry.outbus1","output.inbus1",metal);
          1490             wire ("output.outbus1","outbus1",metal);
E35       1491          END;
          1492
          1493          ! bus 2;
B36       1494          IF twomem THEN BEGIN
          1495             wire ("inbus2","mem(1).inbus2",metal);
          1496             wire ("mem(1).outbus2","mem(2).inbus2",metal);
          1497             wire ("mem(2).outbus2","outbus2",metal);
B37 E36   1498          END ELSE BEGIN
          1499             wire ("inbus2","input.inbus2",metal);
          1500             wire ("input.outbus2","carry.inbus2",metal);
          1501             wire ("carry.outbus2","output.inbus2",metal);
          1502             wire ("output.outbus2","outbus2",metal);
E37       1503          END;
          1504
          1505          ! connect up mem(1) to mem(2);
B38       1506          IF twomem THEN BEGIN
          1507             wire ("mem(1).zoverout","mem(2).zoverin",metal);
          1508             wire ("mem(1).zbaroverout","mem(2).zbaroverin",metal);
```

```
        1509            wire ("mem(1).outdrbus2","mem(2).indrbus2",diff);
        1510
        1511            ! connect mem(2) to input;
        1512            wire ("mem(2).zoverout","zoverout",metal);
        1513            wire ("mem(2).zbaroverout","zbarovercut",metal);
        1514            wire ("mem(2).zout","zout",metal);
        1515            wire ("mem(2),zbarout","zbarout",metal);
E38     1516          END;
        1517
B39     1518          IF NOT twomem THEN BEGIN
        1519            ! CONNECT DATA INPUTS;
        1520            wire ("ain","input.ain",metal);
        1521            wire ("bin","input.bin",metal);
        1522            wire ("abarin","input.abarin",metal);
        1523            wire ("bbarin","input.bbarin",metal);
        1524
        1525            ! connect input to carry;
        1526            wire ("input.kbar","carry.kbar",poly);
        1527            wire ("input.pbar","carry.pbar",poly);
        1528
        1529            ! connect carry to output;
        1530            wire ("carry.pout","output.propin",poly);
        1531            wire ("carry.pbarout","output.pbarin",poly);
        1532            wire ("carry.carout","output.cin",poly);
        1533            wire ("carry.cbarout","output.cinbar",metal);
E39     1534          END;
        1535
B40     1536          IF twomem THEN BEGIN
        1537            ! memory cells;
        1538
B41     1539            FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
        1540              wire (sl("rdmemin1",i),conc(sl("mem",i),".rdinbus1"),poly);
        1541              wire (sl("rdmemout1",i),conc(sl("mem",i),".rdoutbus1"),poly);
        1542              wire (sl("rdmemin2",i),conc(sl("mem",i),".rdinbus2"),poly);
        1543              wire (sl("rdmemout2",i),conc(sl("mem",i),".rdoutbus2"),poly);
        1544              wire (sl("drmemin1",i),conc(sl("mem",i),".drinbus1"),poly);
        1545              wire (sl("drmemout1",i),conc(sl("nem",i),".droutbus1"),poly);
        1546              wire (sl("drmemin2",i),conc(sl("mem",i),".drinbus2"),poly);
        1547              wire (sl("drmemout2",i),conc(sl("nem",i),".droutbus2"),poly);
        1548              wire (sl("refin",i),conc(sl("mem",i),".refin"),poly);
        1549              wire (sl("refout",i),conc(sl("mem',i),".refout"),poly);
        1550              wire (sl("vddinvin",i),conc(sl("mem",i),".vddinvin"),diff);
        1551              wire (sl("vddinvout",i),conc(sl("nem",i),".vddinvout"),diff);
E41     1552            END;
B42 E40 1553          END ELSE BEGIN
        1554
        1555            ! connect input cell pins;
        1556            wire ("pffin","input.pffin",poly);
        1557            wire ("pffout","input.pffout",poly);
        1558            wire ("pttin","input.pttin",poly);
        1559            wire ("pttout","input.pttout",poly);
        1560            wire ("kttin","input.kttin",poly);
```

```
1561            wire ("kttout","input.kttout",poly);
1562            wire ("ktfin","input.ktfin",poly);
1563            wire ("ktfout","input.ktfout",poly);
1564            wire ("ptfin","input.ptfin",poly);
1565            wire ("ptfout","input.ptfout",poly);
1566            wire ("pftin","input.pftin",poly);
1567            wire ("pftout","input.pftout",poly);
1568            wire ("kftin","input.kftin",poly);
1569            wire ("kftout","input.kftout",poly);
1570            wire ("kffin","input.kffin",poly);
1571            wire ("kffout","input.kffout",poly);
1572            wire ("vddinpout","input.vddiffout",diff);
1573            wire ("vddinpin","input.vddiffin",diff);
1574
1575            ! connect carry cell pins;
1576            wire ("vddcarout1","carry.vddiffout1",diff);
1577            wire ("vddcarin1","carry.vddiffin1",diff);
1578            wire ("vddcarout2","carry.vddiffout2",diff);
1579            wire ("vddcarin2","carry.vddiffin2",diff);
1580            wire ("vddcarout3","carry.vddiffout3",diff);
1581            wire ("vddcarin3","carry.vddiffin3",diff);
1582            wire ("cin","carry.cin",poly);
1583            wire ("cout","carry.cout",poly);
1584            wire ("prein","carry.prein",poly);
1585            wire ("preout","carry.preout",poly);
1586
1587            ! connect output cell pins;
1588            wire ("rffin","output.rffin",poly);
1589            wire ("rffout","output.rffout",poly);
1590            wire ("rtfin","output.rtfin",poly);
1591            wire ("rtfout","output.rtfout",poly);
1592            wire ("rftin","output.rftin",poly);
1593            wire ("rftout","output.rftout",poly);
1594            wire ("rttin","output.rttin",poly);
1595            wire ("rttout","output.rttout",poly);
1596            wire ("latchin","output.latchin",poly);
1597            wire ("latchout","output.latchout",poly);
1598            wire ("dr1resin","output.drive1in",poly);
1599            wire ("dr1resout","output.drive1out",poly);
1600            wire ("dr2resin","output.drive2in",poly);
1601            wire ("dr2resout","output.drive2out",poly);
E42     1602    END;
1603
1604    !*************************************************************;
1605    ! place pins physically;
1606    !*************************************************************;
1607
1608    ! horizontal busses;
1609    IF twomem THEN place ("vddin",position ("mem(1).vddin"))
1610    ELSE place ("vddin",position ("input.vddin"));
1611    IF NOT twomem THEN place ("vddout",position ("output.vddout"))
1612    ELSE place ("vddout",position ("mem(2).vddout"));
```

```
        1613            IF twomem THEN place ("gndin",position ("mem(1).gndin"))
        1614            ELSE place ("gndin",position ("input.gndin"));
        1615            IF NOT twomem THEN place ("gndout",position ("output.gndout"))
        1616            ELSE place ("gndout",position ("mem(2).gndout"));
        1617            IF twomem THEN place ("inbus1",position ("mem(1).inbus1"))
        1618            ELSE place ("inbus1",position ("input.irbus1"));
        1619            IF NOT twomem THEN place ("outbus1",position ("output.outbus1"))
        1620            ELSE place ("outbus1",position ("mem(2).outbus1"));
        1621            IF twomem THEN place ("inbus2",position ("mem(1).inbus2"))
        1622            ELSE place ("inbus2",position ("input.irbus2"));
        1623            IF NOT twomem THEN place ("outbus2",position ("output.outbus2"))
        1624            ELSE place ("outbus2",position ("mem(2).outbus2"));
        1625
B43     1626            IF twomem THEN BEGIN
        1627               place ("zoverout",position ("mem(2).zoverout"));
        1628               place ("zbaroverout",position ("mem(2).zbaroverout"));
        1629               place ("zout",position ("mem(2).zout"));
        1630               place ("zbarout",position ("mem(2).zbarout"));
B44 E43 1631            END ELSE BEGIN
        1632               place ("ain",position ("INPUT.AIN"));
        1633               place ("bin",position ("input.bin"));
        1634               place ("abarin",position ("input.abarin"));
        1635               place ("bbarin",position ("input.bbarin"));
E44     1636            END;
        1637
        1638            !vertical control lines and power fixes;
        1639
B45     1640            IF twomem THEN BEGIN
        1641               ! memory cells;
        1642
B46     1643               FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
        1644                  place (sl("rdmemin1",i),position
        1645                  (conc(sl("mem",i),".rdinbus1")));
        1646                  place (sl("rdmemout1",i),position
        1647                  (conc(sl("mem",i),".rdoutbus1")));
        1648                  place (sl("rdmemin2",i),position
        1649                  (conc(sl("mem",i),".rdinbus2")));
        1650                  place (sl("rdmemout2",i),position
        1651                  (conc(sl("mem",i),".rdoutbus2")));
        1652                  place (sl("drmemin1",i),position
        1653                  (conc(sl("mem",i),".drinbus1")));
        1654                  place (sl("drmemout1",i),position
        1655                  (conc(sl("mem",i),".droutbus1")));
        1656                  place (sl("drmemin2",i),position
        1657                  (conc(sl("mem",i),".drinbus2")));
        1658                  place (sl("drmemout2",i),position
        1659                  (conc(sl("mem",i),".droutbus2")));
        1660                  place (sl("refin",i),position (conc(sl("mem",i),".refin")));
        1661                  place (sl("refout",i),position (conc(sl("mem",i),".refout")));
        1662                  place (sl("vddinvin",i),position
        1663                  (conc(sl("mem",i),".vddinvin")));
        1664                  place (sl("vddinvout",i),position
```

```
        1665                (conc(sl("mem",i),".vddinvout")));
E46     1666             END;
B47 E45 1667          END ELSE BEGIN
        1668
        1669             ! place input cell pins;
        1670             place ("pffin",position ("input.pffin"));
        1671             place ("pffout",position ("input.pffout"));
        1672             place ("pttin",position ("input.pttin"));
        1673             place ("pttout",position ("input.pttout"));
        1674             place ("kttin",position ("input.kttin"));
        1675             place ("kttout",position ("input.kttout"));
        1676             place ("ktfin",position ("input.ktfin"));
        1677             place ("ktfout",position ("input.ktfout"));
        1678             place ("ptfin",position ("input.ptfin"));
        1679             place ("ptfout",position ("input.ptfout"));
        1680             place ("pftin",position ("input.pftin"));
        1681             place ("pftout",position ("input.pftout"));
        1682             place ("kftin",position ("input.kftin"));
        1683             place ("kftout",position ("input.kftout"));
        1684             place ("kffin",position ("input.kffin"));
        1685             place ("kffout",position ("input.kffout"));
        1686             place ("vddinpout",position ("input.vddiffout"));
        1687             place ("vddinpin",position ("input.vddiffin"));
        1688
        1689             ! place carry cell pins;
        1690             place ("vddcarout1",position ("carry.vddiffout1"));
        1691             place ("vddcarin1",position ("carry.vddiffin1"));
        1692             place ("vddcarout2",position ("carry.vddiffout2"));
        1693             place ("vddcarin2",position ("carry.vddiffin2"));
        1694             place ("vddcarout3",position ("carry.vddiffout3"));
        1695             place ("vddcarin3",position ("carry.vddiffin3"));
        1696             place ("cin",position ("carry.cin"));
        1697             place ("cout",position ("carry.cout"));
        1698             place ("prein",position ("carry.prein"));
        1699             place ("preout",position ("carry.preout"));
        1700
        1701             ! place output cell pins;
        1702             place ("rffin",position ("output.rffin"));
        1703             place ("rffout",position ("output.rffout"));
        1704             place ("rtfin",position ("output.rtfin"));
        1705             place ("rtfout",position ("output.rtfout"));
        1706             place ("rftin",position ("output.rftin"));
        1707             place ("rftout",position ("output.rftout"));
        1708             place ("rttin",position ("output.rttin"));
        1709             place ("rttout",position ("output.rttout"));
        1710             place ("latchin",position ("output.latchin"));
        1711             place ("latchout",position ("output.latchout"));
        1712             place ("dr1resin",position ("output.drive1in"));
        1713             place ("dr1resout",position ("output.drive1out"));
        1714             place ("dr2resin",position ("output.drive2in"));
        1715             place ("dr2resout",position ("output.drive2out"));
E47     1716          END;
```

```
E23     1717        END of om2bitslice;
        1718
        1719        !*************************************************************;
        1720        !      array of OM2 bit slices                                ;
        1721        !*************************************************************;
        1722
        1723        blockdef CLASS om2aludef (yrep,slice,twomem,type);
        1724        VALUE yrep,type; INTEGER yrep; TEXT type; REF (om2bitslice) slice;
        1725        BOOLEAN twomem;
B48     1726        BEGIN
        1727           INTEGER i,j;
        1728           TEXT lastslice,lastslicedot;
        1729           REF (point) vddinc;
        1730
        1731           ! set text name of cell definition;
        1732           nameblock (conc ("om2alu.",type));
        1733
        1734           ! set block bounding box;
        1735           blockboundbox (0,0, slice.xlim,ylim*yrep);
        1736
        1737           !*****************************************************;
        1738           ! pin specification                                   ;
        1739           !*****************************************************;
        1740
        1741           ! horizontal;
B49     1742           FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
        1743              pin (sl("vddin",i)); pin (sl("vddout',i)); pin (sl("gndin",i));
        1744              pin (sl('gndout",i)); pin (sl("inbus1",i)); pin (sl("outbus1",i));
        1745              pin (sl("inbus2",i)); pin (sl("outbus2",i));
E49     1746           END;
        1747
        1748           ! extra vdd line at top;
        1749           pin (sl("vddin",yrep+1)); pin (sl("vddout",yrep+1));
        1750
        1751           ! vertical pins - as for bit slice;
B50     1752           IF twomem THEN BEGIN
        1753              ! memory cells 1 and 2;
B51     1754              FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
        1755                 pin (sl ("rdmemin1",i)); pin (sl('rdmemout1",i));
        1756                 pin (sl("rdmemin2",i)); pin (sl("rdmemout2",i));
        1757                 pin (sl("drmemin1",i)); pin (sl("drmemout1",i));
        1758                 pin (sl("drmemin2",i)); pin (sl("drmemout2",i));
        1759                 pin (sl("refin",i)); pin (sl("refout",i));
        1760                 pin (sl("vddinvin",i)); pin (sl("vddinvout",i));
E51     1761              END;
        1762
B52 E50 1763           END ELSE BEGIN
        1764
        1765              ! input cell;
        1766              pin ("pffin"); pin ("pffout"); pin ('pttin"); pin ("pttout");
        1767              pin ("kttin"); pin ("kttout"); pin ('ktfin"); pin ("ktfout");
        1768              pin ("ptfin"); pin ("ptfout"); pin ('pftin"); pin ("pftout");
```

```
        1769            pin ("kftin"); pin ("kftout"); pin ("kffin"); pin ("kffout");
        1770            pin ("vddinpout"); pin ("vddinpin");
        1771
        1772            ! carry cell;
        1773            pin ("vddcarout1"); pin ("vddcarin1"); pin ("vddcarout2");
        1774            pin ("vddcarin2");
        1775            pin ("vddcarout3"); pin ("vddcarin3");
        1776            pin ("cin"); pin ("cout"); pin ("prein"); pin ("preout");
        1777
        1778            ! output cell;
        1779            pin ("rffin"); pin ("rffout"); pin ("rtfin"); pin ("rtfout");
        1780            pin ("rftin"); pin ("rftout"); pin ("rttin"); pin ("rttout");
        1781            pin ("latchin"); pin ("latchout");
        1782            pin ("dr1resin"); pin ("dr1resout"); pin ("dr2resin");
        1783            pin ("dr2resout");
E52     1784          END;
        1785
        1786          !*!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!*!!*!!*!*!!*!!*!!*!!*!!*;
        1787          !                  alu bit slice instncing                    ;
        1788          !!*!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!!*!;
        1789
        1790          FOR i:=1 STEP 1 UNTIL yrep DO
        1791          blockinst (sl ("slice",i),slice, NEW transform (NONE).
        1792          translatedby (NEW point (0,(i-1)*ylim)));
        1793
        1794          lastslice:-Copy (sl("slice",yrep));
        1795          lastslicedot:-Copy (conc(lastslice,"."));
        1796
        1797          ! wire in top vdd line;
B53     1798          IF twomem THEN BEGIN
        1799            wire (sl("vddin",yrep+1),"memvx1",metal).width(4);
        1800            wire ("memvx1",conc(lastslice,".vddinvout(1)"),diff);
        1801            wire ("memvx1","memvx2",metal).width(4);
        1802            wire ("memvx2",conc(lastslice,".vddinvout(2)"),diff);
        1803            wire ("memvx2",sl("vddout",yrep+1),metal).width(4);
B54 E53 1804          END ELSE BEGIN
        1805            wire (sl("vddin",yrep+1),"inpvx",metal);
        1806            wire ("inpvx",conc(lastslice,".vddinpin"),diff);
        1807            wire ("inpvx","carvx1",metal).width(4);
        1808            wire ("carvx1",conc(lastslice,".vddcarin1"),diff);
        1809            wire ("carvx1","carvx2",metal).width(4);
        1810            wire ("carvx2",conc(lastslice,".vddcarin2"),diff);
        1811            wire ("carvx2","carvx3",metal).width(4);
        1812            wire ("carvx3",conc(lastslice,".vddcarin3"),diff);
        1813            wire ("carvx3",sl("vddout",yrep+1),metal).width(4);
E54     1814          END;
        1815
        1816          ! vertical bussing - control lines to pins;
        1817
        1818          ! memory;
B55     1819          IF twomem THEN BEGIN
B56     1820            FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
```

```
1821                wire (sl("rdmemin1",i),conc("slice(1).",sl("rdmemin1",i)),poly);
1822                wire (sl("rdmemin2",i),conc("slice(1).",sl("rdmemin2",i)),poly);
1823                wire (sl("drmemin1",i),conc("slice(1).",sl("drmemin1",i)),poly);
1824                wire (sl("drmemin2",i),conc("slice(1).",sl("drmemin2",i)),poly);
1825                wire (sl("refin",i),conc("slice(1).",sl("refin",i)),poly);
1826
1827                wire
1828                (sl("rdmemout1",i),conc(lastslicedot,sl("rdmemout1",i)),poly);
1829                wire
1830                (sl("rdmemout2",i),conc(lastslicedot,sl("rdmemout2",i)),poly);
1831                wire
1832                (sl("drmemout1",i),conc(lastslicedot,sl("drmemout1",i)),poly);
1833                wire
1834                (sl("drmemout2",i),conc(lastslicedot,sl("drmemout2",i)),poly);
1835                wire (sl("refout",i),conc(lastslicedot,sl("refout",i)),poly);
E56             1836             END;
B57 E55         1837          END ELSE BEGIN
1838
1839             ! input;
1840             wire ("pffin","slice(1).pffin",poly);
1841             wire ("pttin","slice(1).pttin",poly);
1842             wire ("kttin","slice(1).kttin",poly);
1843             wire ("ktfin","slice(1).ktfin",poly);
1844             wire ("ptfin","slice(1).ptfin",poly);
1845             wire ("pftin","slice(1).pftin",poly);
1846             wire ("kftin","slice(1).kftin",poly);
1847             wire ("kffin","slice(1).kffin",poly);
1848
1849             wire ("pffout",conc(lastslice,"pffout'),poly);
1850             wire ("pttout",conc(lastslice,"pttout'),poly);
1851             wire ("kttout",conc(lastslice,"kttout'),poly);
1852             wire ("ktfout",conc(lastslice,"ktfout'),poly);
1853             wire ("ptfout",conc(lastslice,"ptfout'),poly);
1854             wire ("pftout",conc(lastslice,"pftout'),poly);
1855             wire ("kftout",conc(lastslice,"kftout'),poly);
1856             wire ("kffout",conc(lastslice,"kffout'),poly);
1857
1858             ! carry;
1859             wire ("cin","slice(1).cin",diff);
1860             wire ("cout",conc(lastslice,".cout"),diff);
1861             wire ("prein","slice.prein",poly);
1862             wire ("preout",conc(lastslice,"preout'),poly);
1863
1864             ! output;
1865             wire ("rffin","slice(1).rffin",poly);
1866             wire ("rtfin","slice(1).rtfin",poly);
1867             wire ("rftin","slice(1).rftin",poly);
1868             wire ("rttin","slice(1).rttin",poly);
1869             wire ("latchin","slice(1).latchin",poly);
1870             wire ("dr1resin","slice(1).dr1resin",poly);
1871             wire ("dr2resin","slice(1).dr2resin",poly);
1872
```

```
                1873            wire ("rffout",conc(lastslice,".rffout"),poly);
                1874            wire ("rtfout",conc(lastslice,".rtfout"),poly);
                1875            wire ("rftout",conc(lastslice,".rftout"),poly);
                1876            wire ("rttout",conc(lastslice,".rttout"),poly);
                1877            wire ("latchout",conc(lastslice,".latchout"),poly);
                1878            wire ("dr1resout",conc(lastslice,".dr1resout"),poly);
                1879            wire ("dr2resout",conc(lastslice,".dr2resout"),poly);
E57             1880          END;
                1881
                1882          ! connect up internal vertical control lines;
B58             1883          FOR i:=1 STEP 1 UNTIL yrep-1 DO BEGIN
                1884
                1885            ! memory;
B59             1886            IF twomem THEN BEGIN
                1887
B60             1888              FOR j:=1 STEP 1 UNTIL 2 DO BEGIN
                1889
                1890                wire (conc(sl("slice",i),".",sl("rdmemout1",j)),
                1891                conc(sl("slice",i+1),".",sl("rdmemin1",j)),poly);
                1892                wire (conc(sl("slice",i),".",sl("rdmemout2",j)),
                1893                conc(sl("slice",i+1),".",sl("rdmemin2",j)),poly);
                1894                wire (conc(sl("slice",i),".",sl("drmemout1",j)),
                1895                conc(sl("slice",i+1),".",sl("dmemin1",j)),poly);
                1896                wire (conc(sl("slice",i),".",sl("drmemout2",j)),
                1897                conc(sl("slice",i+1),".",sl("dmemin2",j)),poly);
                1898                wire (conc(sl("slice",i),".",sl("refout",j)),
                1899                conc(sl("slice",i+1),".",sl("refin",j)),poly);
E60             1900              END;
B61 E59         1901            END ELSE BEGIN
                1902
                1903              ! input;
                1904
                1905              wire
                1906              (conc(sl("slice",i),".pffout"),conc(sl("slice",i+1),'.pffin"),
                1907              poly);
                1908              wire
                1909              (conc(sl("slice",i),".pttout"),conc(sl("slice",i+1),'.pttin"),
                1910              poly);
                1911              wire
                1912              (conc(sl("slice",i),".kttout"),conc(sl("slice",i+1),'.kttin"),
                1913              poly);
                1914              wire
                1915              (conc(sl("slice",i),".ktfout"),conc(sl("slice",i+1),'.ktfin"),
                1916              poly);
                1917              wire
                1918              (conc(sl("slice",i),".ptfout"),conc(sl("slice",i+1),'.ptfin"),
                1919              poly);
                1920              wire
                1921              (conc(sl("slice",i),".pftout"),conc(sl("slice",i+1),'.pftin"),
                1922              poly);
                1923              wire
                1924              (conc(sl("slice",i),".kftout"),conc(sl("slice",i+1),'.kftin"),
```

```
                    poly);
                    wire
                    (conc(sl("slice",i),".kffout"),conc(sl("slice",i+1),".kffin"),
                    poly);

                    ! carry;
                    wire
                    (conc(sl("slice",i),".cout"),conc(sl("slice",i+1),".cin"),diff);
                    wire
                    (conc(sl("slice",i),".preout"),conc(sl("slice",i+1),".prein"),
                    poly);

                    ! output;
                    wire
                    (conc(sl("slice",i),".rffout"),conc(sl("slice",i+1),".rffin"),
                    poly);
                    wire
                    (conc(sl("slice",i),".rtfout"),conc(sl("slice",i+1),".rtfin"),
                    poly);
                    wire
                    (conc(sl("slice",i),".rftout"),conc(sl("slice",i+1),".rftin"),
                    poly);
                    wire
                    (conc(sl("slice",i),".rttout"),conc(sl("slice",i+1),".rttin"),
                    poly);
                    wire (conc(sl("slice",i),".latchout"),conc(sl("slice",i+1),
                    ".latchin"),poly);
                    wire (conc(sl("slice",i),".dr1resout"),conc(sl("slice',i+1),
                    ".dr1resin"),poly);
                    wire (conc(sl("slice",i),".dr2resout"),conc(sl("slice',i+1),
                    ".dr2resin"),poly);
B61                 END;
B58               END;

                  ! put in contacts;
B62               IF twomem THEN BEGIN
                     contact ("memvx1"); contact ("memvx2");
B63 E62           END ELSE BEGIN
                     contact ("inpvx");
                     contact ("carvx1"); contact ("carvx2"); contact ("carvx3");
E63               END;

                  !*********************************************************;
                  ! place pins physically;
                  !*********************************************************;

                  ! horizontal busses;
B64               FOR i:=1 STEP 1 UNTIL yrep DO BEGIN
                     place (sl("vddin",i),position (conc(sl("slice",i),".vddin")));
                     place (sl("vddout",i),position (conc(sl("slice",i),".vddout")));
                     place (sl("gndin",i),position (conc(sl("slice",i),".gndin")));
                     place (sl("gndout",i),position (conc(sl("slice",i),".gndout")));
```

```
                1977            place (sl("inbus1",l),position (conc(sl("slice",i),".inbus1")));
                1978            place (sl("outbus1",i),position (conc(sl("slice",i),".outbus1")));
                1979            place (sl("inbus2",i),position (conc(sl("slice",i),".inbus2")));
                1980            place (sl("outbus2",i),position (conc(sl("slice",i),".outbus2")));
E64             1981         END;
                1982         place (sl("vddin",yrep+1),NEW point (0,ylim*yrep+2));
                1983         place (sl("vddout",yrep+1),NEW point (slice.xlim,ylim*yrep+2));
                1984
                1985         !vertical control lines and power fixes;
                1986
                1987         ! memory cells;
B65             1988         IF twomem THEN BEGIN
                1989
B66             1990            FOR i:=1 STEP 1 UNTIL 2 DO BEGIN
                1991               place (sl("rdmemin1",i),position
                1992               (conc("slice(1).",sl("rdmemin1",i))));
                1993               place (sl("rdmemout1",i),position (conc(lastslice,
                1994               ".",sl("rdmemout1",i))));
                1995               place (sl("rdmemin2",i),position
                1996               (conc("slice(1).",sl("rdmemin2",i))));
                1997               place (sl("rdmemout2",i),position (conc(lastslicedot,
                1998               sl("rdmemout2",i))));
                1999               place (sl("drmemin1",i),position
                2000               (conc("slice(1).",sl("drmemin1",i))));
                2001               place (sl("drmemout1",i),position (conc(lastslicedot,
                2002               sl("drmemout1",i))));
                2003               place (sl("drmemin2",i),position
                2004               (conc("slice(1).",sl("drmemin2",i))));
                2005               place (sl("drmemout2",i),position (conc(lastslicedot,
                2006               sl("drmemout2",i))));
                2007               place (sl("refin",i),position (conc("slice(1).",sl("refin",i))));
                2008               place (sl("refout",i),position (conc(lastslicedot,
                2009               sl("refout",i))));
                2010               place (sl("vddinvin",i),position
                2011               (conc("slice(1).",sl("vddinvin",i))));
                2012               place (sl("vddinvout",i),position (conc(lastslicedot,
                2013               sl("vddinvout",i))));
E66             2014            END;
                2015
B67 E65         2016         END ELSE BEGIN
                2017            ! place input cell pins;
                2018            place ("pffin",position ("slice(1).pffin"));
                2019            place ("pffout",position (conc(lastslice,".pffout")));
                2020            place ("pttin",position ("slice(1).pttin"));
                2021            place ("pttout",position (conc(lastslice,".pttout")));
                2022            place ("kttin",position ("slice(1).kttin"));
                2023            place ("kttout",position (conc(lastslice,".kttout")));
                2024            place ("ktfin",position ("slice(1).ktfin"));
                2025            place ("ktfout",position (conc(lastslice,".ktfout")));
                2026            place ("ptfin",position ("slice(1).ptfin"));
                2027            place ("ptfout",position (conc(lastslice,".ptfout")));
                2028            place ("pftin",position ("slice(1).pftin"));
```

```
2029            place ("pftout",position (conc(lastslice,".pftout")));
2030            place ("kftin",position ("slice(1).kftin"));
2031            place ("kftout",position (conc(lastslice,".kftout")));
2032            place ("kffin",position ("slice(1).kffin"));
2033            place ("kffout",position (conc(lastslice,".kffout")));
2034            place ("vddinpout",position ("slice(1).vddinpout"));
2035            place ("vddinpin",position (conc(lastslice,".vddinpin")));
2036
2037            ! place carry cell pins;
2038            place ("vddcarout1",position ("slice(1).vddcarout1"));
2039            place ("vddcarin1",position (conc(lastslice,".vddcarin1")));
2040            place ("vddcarout2",position ("slice(1).vddcarout2"));
2041            place ("vddcarin2",position (conc(lastslice,".vddcarin2")));
2042            place ("vddcarout3",position ("slice(1).vddcarout3"));
2043            place ("vddcarin3",position (conc(lastslice,".vddcarin3")));
2044            place ("cin",position ("slice(1).cin"));
2045            place ("cout",position (conc(lastslice,".cout")));
2046            place ("prein",position ("slice(1).prein"));
2047            place ("preout",position (conc(lastslice,".preout")));
2048
2049            ! place output cell pins;
2050            place ("rffin",position ("slice(1).rffin"));
2051            place ("rffout",position (conc(lastslice,".rffout")));
2052            place ("rtfin",position ("slice(1).rtfin"));
2053            place ("rtfout",position (conc(lastslice,".rtfout")));
2054            place ("rftin",position ("slice(1).rftin"));
2055            place ("rftout",position (conc(lastslice,".rftout")));
2056            place ("rttin",position ("slice(1).rttin"));
2057            place ("rttout",position (conc(lastslice,".rttout")));
2058            place ("latchin",position ("slice(1).latchin"));
2059            place ("latchout",position (conc(lastslice,".latchout")));
2060            place ("dr1resin",position ("slice(1).drivelin"));
2061            place ("dr1resout",position (conc(lastslice,".drivelout")));
2062            place ("dr2resin",position ("slice(1).drive2in"));
2063            place ("dr2resout",position (conc(lastslice,".drive2out")));
E67 2064        END;
2065
2066        ! place power rail contacts;
2067        vddinc:-NEW point (0,2);
B68 2068    IF twomem THEN BEGIN
2069            place ("memvx1",position (conc
2070            (lastslice,".vddinvout(1)")).plus(vddinc));
2071            place
2072            ("memvx2",position(conc(lastslice,".vddinvout(2)")).plus(vddinc));
B69 E68 2073 END ELSE BEGIN
2074            place ("inpvx",position (conc(lastslice,".vddinpin")).plus(vddinc));
2075            place ("carvx1",position
2076            (conc(lastslice,".vddcarin1")).plus(vddinc));
2077            place ("carvx2",position
2078            (conc(lastslice,".vddcarin2")).plus(vddinc));
2079            place ("carvx3",position
2080            (conc(lastslice,"vddcarin3")).plus(vddinc));
```

```
E69     2081            END;
        2082
E48     2083        END of om2blockdef;
        2084
        2085        vdd:=2; gnd:=40; bus1:=66.5; bus2:=10.5; ylim:=73;
        2086        initdefnopt; initsimopt; mainoptions (defnoptions,simoptions);
E2      2087      END;
E1      2088    END of program;


SWITCHES CHANGED FROM DEFAULT:

  -W NO WARNINGS GENERATED

ERRORS DETECTED:
    7 TYPE W MESSAGES
```